SRC TR 87-164

TECHNICAL RESEARCH REPORT

Control System Design for a Flexible Arm

by

L.-S. Wang

SYSTEMS RESEARCH CENTER

UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND 20742

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1987		**00-00-1987 to 00-00-1987**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Control System Design for a Flexible Arm	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Maryland,The Graduate School,2123 Lee Building,College Park,MD,20742**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**92**	

ABSTRACT

Title of Thesis: Control System Design for a Flexible Arm

Li-Sheng Wang, Master of Science, 1987

Thesis directed by: Dr. P. S. Krishnaprasad
Professor
Department of Electrical Engineering

In this thesis, we study the problem of real-time control of a flexible arm. We have investigated techniques for compensating the effects of friction and ripple torque. New software was written to use a Metrabyte Data Acquisition and Control board for the real time implementation. A controller-observer scheme was used together with integral feedback. In the design of feedback gains, a newly developed package called CONSOLE was used. After translating the continuous-time design to the discrete-system and before implementation, a package called SIMNON was used to do the simulation of the whole system and to explore the effect of different sampling rates. The experiments done so far imply that the schemes used here are sound for real-time control of flexible structures.

CONTROL SYSTEM DESIGN FOR A FLEXIBLE ARM

By

Li-Sheng Wang

Thesis submitted to the faculty of the Graduate School
of University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1987

Advisory Committee :

Professor: Dr. P. S. Krishnaprasad
Associate professor: Dr. André L. Tits
Associate professor: Dr. Eyad Abed

ACKNOWLEDGEMENTS

I owe much thanks to many people who helped me with the experiment and in developing this thesis. First, I would like to express my sincerest appreciation to my advisor Professor P.S. Krishnaprasad for his kind guidance and patience during the whole period of my work. Next, I thank Mr. Haven Frank from Westinghouse for establishment and maintenance of the hardware system and many helpful ideas. I appreciate the instruction of Dr. M.K.H. Fan in developing and using the software package CONSOLE . Thanks are also due to Mr. Buno Pati, for helping me in designing the filter and handshaking scheme; to Mr. Kevin Anderson, for discussing with me some problems in using Microsoft C; to Mr. Shyam Mehrotra of the Electrical Engineering Laboratories, for his assistance in obtaining electronic components; to Ms. Tien-Chun Wang, for drawing the picture of the hardware system. In addition, I gratefully acknowledge the generous fellowship support of the Systems Research Center at the University of Maryland, and the support of the University Research Initiative Program of the AFOSR. They made the work possible. Finally, I thank my wife, Yuh-Yin, for helping me with the typing as well as putting up with me for so long a time.

TABLE OF CONTENTS

CHAPTER I

Introduction

As noted in many papers(e.g. [19]) , light-weight low-inertia links will be widely used in the next generation of robot manipulators. But as the inertia decreases, some new problems will arise. For example, with a DC motor direct-drive system, the effects of ripple torque becomes significant as the load inertia decreases. Here, following the experiment of Frank[11], a computer-controlled system with a light flexible arm is studied. After illustrating the experimental setup of the whole system (Chapter 2) and deriving the beam equations under specific assumptions (chapter 3), certain general steps for designing a computer-controlled system are used throughout this thesis. They are:

1. Develop a mathematical model.
2. Design a suitable controller structure with some free parameters.
3. By using optimization packages with simulators, e.g. CONSOLE and *MaryLin* , obtain the optimized design to meet the specifications desired.
4. Translate the continuous-time controller to the discrete-time case.
5. By using a hybrid simulator, e.g. SIMNON , simulate the whole computer-controlled system and choose a suitable sampling rate.
6. Implement the controller in the real-time system.

With this sequence of steps and some tricks in real-time programming, a computer-controlled system can be designed successfully.

In the following chapters, the controller for a flexible arm is first designed by CONSOLE with *MaryLin* in Chapter 4 to show how to use that powerful package. In Chapter 5, a new idea about integral control is suggested and is shown by a simple example. The characteristics of the actuator is discussed in Chapter 6. They include friction modeling and ripple torque in a DC brush motor system. One way to compensate their effects has also been suggested. Chapter 7 describes the problems which were met in implementation and ways to handle them. After discussing the

details of the experiment, the ideas in Chapter 5 and 6 are included in the design of the controller for the flexible arm and implemented in the real-time system. The experiment shows that a combination of these ideas leads to notable performance improvement. The last chapter(8) presents the conclusions and outlines the future work.

CHAPTER II

Experimental Setup

2.1 Environment

The system under consideration is shown in Fig. 2-1.

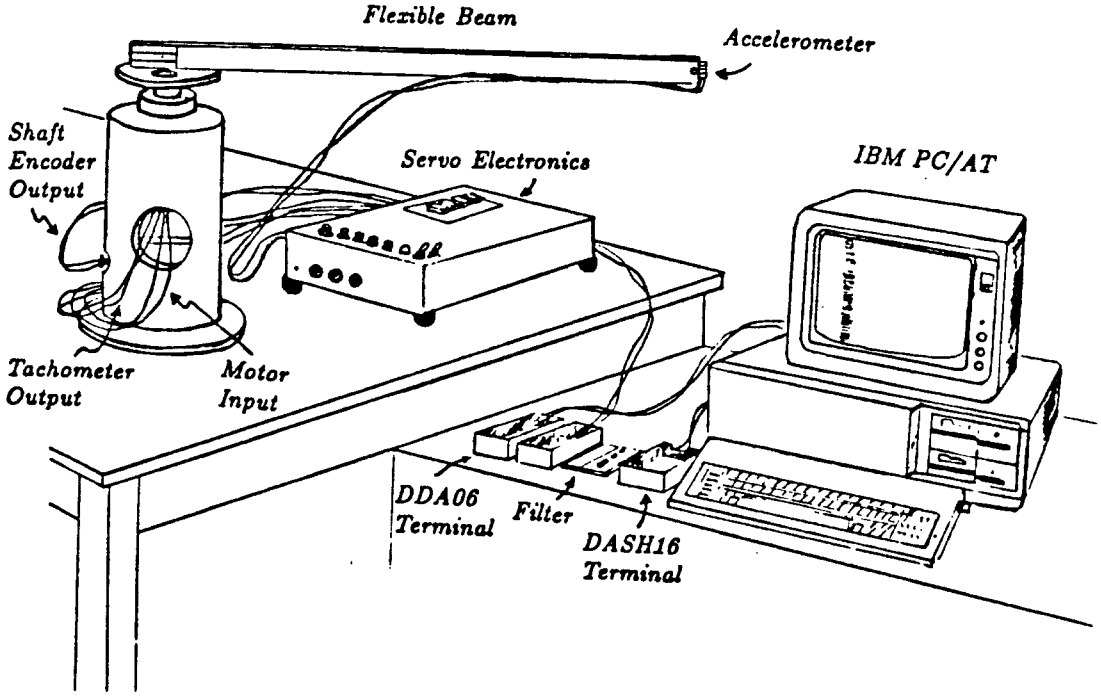


Figure 2-1 System under Consideration

It includes:

1. a 1 meter flexible beam.
2. a DC brush motor
3. three sensors: position encoder, tachometer, and accelerometer.
4. a Metrabyte DDA06 A/D & D/A board

5. a Metrabyte DASH16 A/D & D/A board
6. analog filters and voltage buffers
7. IBM PC/AT

By way of A/D and D/A converter and digital I/O, the software on IBM PC/AT can then control the system according to some specific control laws. The details about the implementation will be discussed in Chapter 7. In the next section, the computer-to-motor model (without flexible beam) will be established.

2.2 Computer-to-Motor Model

The system block diagram from computer output to motor position may be modeled as in Fig. 2-2.

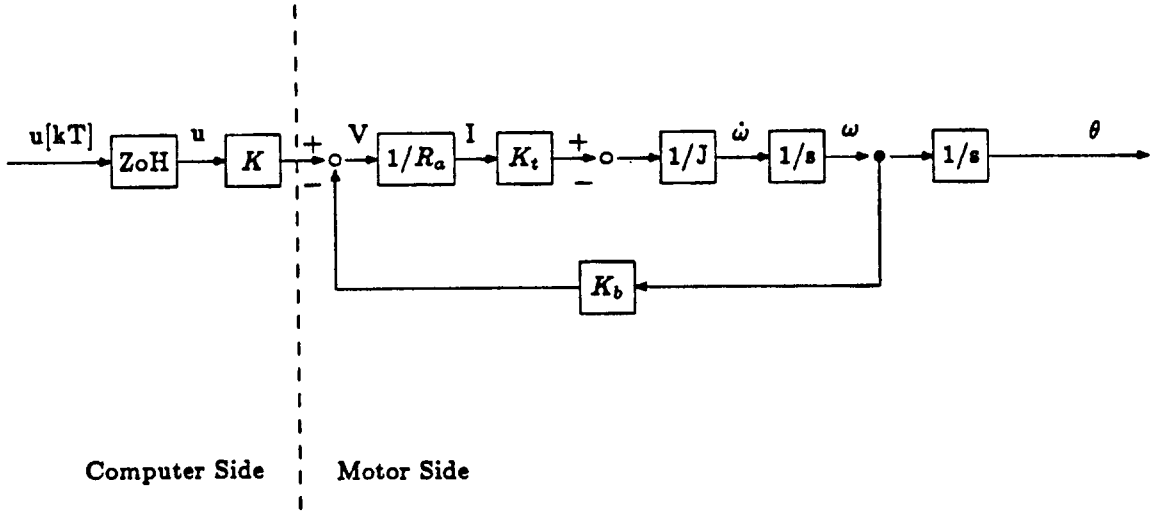


Fig. 2-2 Block Diagram of the Computer-to-Motor System

with the blocks:

ZoH : Zero-Order Hold
K : Power Amplifier Gain
R_a : Motor Armature Resistance
K_b : Motor Back EMF Constant
K_t : Motor Torque Constant
J : Inertia

By the input u[kT] from IBM PC/AT, the motor can be controlled in some specific way. The state equation of the whole system can be represented as

$$\begin{aligned} J\dot{\omega}(t) &= T(t) \\ &= \frac{K_t}{R_a} V(t) \\ &= -\frac{K_t K_b}{R_a}\omega(t) + \frac{K K_t}{R_a} u(t) \end{aligned} \tag{2.1}$$

$$\dot{\theta}(t) = \omega(t) \tag{2.2}$$

In order to get the actual values of the motor parameters, it is necessary to set up an experiment for identifying them. Our approach of determining the motor back emf constant (K_b) is to use another drive (e.g. a tachometer) to drive the motor and measuring the voltage across the motor armature (i.e. let the motor be run as a generator.) Then K_b is the ratio of the voltage and the angular velocity. The angular velocity may be measured by the position encoder and the time counter on the DASH16 board.

After K_b has been found, the motor torque constant K_t can be calculated by the formula, as discussed in [8], corresponding to the specific unit,

$$K_t \text{ (with the unit of } lb-in/amp) = 8.844\ K_b \text{ (with the unit of } Volt/rad/sec)$$

From the experiment, these parameters are (nominal value), respectively,

K = 6.02

K_b = 2.443 (Volt/rad/sec)

K_t = 21.62 (lb-in/amp)

R_a = 33.6 (Ω)

J : depends on the load of the motor

This model will be used in Chapter 5 and 6 to demonstrate the usefulness of some compensators.

CHAPTER III

Mathematical Modeling

In this chapter, the mathematical modeling of the flexible arm will be discussed. After introducing some basic principles in the mechanics and the calculus of variations, three different beam equations are derived.

3.1 Basic Principles

First of all, Hamilton's principle is introduced.

Hamilton's Principle

Assume all forces (besides the constraint forces) are derivable from a generalized scalar potential (may be a function of coordinates, velocities, and time), i.e. the system is monogenic. Then, the motion of the system from time t_1 to t_2 is such that the line integral

$$I = \int_{t_1}^{t_2} L\, dt,$$

where $L = T - V$ where L is the *Lagrangian*, T is the *kinetic energy*, and V is the *potential energy*, has a stationary value for the correct path of the motion, i.e. the variation of I

$$\delta I = \delta \int_{t_1}^{t_2} L(q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n, t)\, dt = 0. \tag{3.1}$$

∎

Here, "stationary" means that the integral I has the same value (to within the first-order infinitesimal) as that integral taken along all neighboring paths.

It can be shown [14] that Lagrange's Equations

$$\frac{\partial f}{\partial q_i} - \frac{d}{dt}\frac{\partial f}{\partial \dot{q}_i} = 0 \quad i = 1, \ldots, n$$

for mechanics of systems of particles imply Hamilton's principle, and, conversely, under the assumption that the system constraint is holonomic (e.g. rigid body), Hamilton's principle also implies the Lagrange's equation.

Next, some basic facts in the calculus of variations are introduced. The proofs of these lemmas can be found in [15].

Lemma 3.1

If x_1, $x_2 (> x_1)$ are constants and $G(x)$ is continuous for $x_1 \leq x \leq x_2$, and if

$$\int_{x_1}^{x_2} \eta(x) G(x)\, dx = 0$$

for every choice of the continuously differentiable function $\eta(x)$ for which $\eta(x_1) = \eta(x_2) = 0$, then

$$G(x) = 0, \quad \forall x_1 \leq x \leq x_2.$$

∎

The above lemma can be extended to the case of double integral.

Lemma 3.2

If D is a domain of the x-y plane and

$$\iint_D \eta(x,y) G(x,y)\, dx\, dy = 0$$

for every continuously differentiable function η such that

$$\eta(x,y) = 0 \quad \forall (x,y) \in \partial D \quad \text{(boundary of } D\text{)}$$

then

$$G(x,y) = 0, \quad \forall (x,y) \in D.$$

∎

Using the above lemmas, the condition for the extremum of some integrals can be derived. For example, the Euler-Lagrange Differential Equation can be derived by explicitly computing the variation δI and invoking lemmas 3.1 and 3.2.

For the case of one independent parameter with one variable, the necessary condition for the extremum of I, where

$$I = \int_{t_1}^{t_2} f(t, y, y')\, dt,$$

is

$$\frac{\partial f}{\partial y} - \frac{d}{dt}\frac{\partial f}{\partial y'} = 0. \tag{3.2}$$

If there are two independent parameters, the necessary condition for the extremum of I, with now

$$I = \iint_D f(x, t, y, y_{,x}, y_{,t})\, dx\, dt,$$

where $y_{,x} = \frac{\partial y}{\partial x}$ and $y_{,t} = \frac{\partial y}{\partial t}$ (this convention will be used in the subsequent sections, the comma is reserved for the case where there is an index with a variable), is

$$\frac{\partial f}{\partial y} - \frac{\partial}{\partial x}\frac{\partial f}{\partial y_{,x}} - \frac{\partial}{\partial t}\frac{\partial f}{\partial y_{,t}} = 0. \tag{3.3}$$

In general, for the n–fold integral

$$I = \int \cdots \int_D f(x_1, \ldots, x_n, w, w_{,x_1}, \ldots, w_{,x_n})\, dx_1 \cdots dx_n,$$

the necessary condition is

$$\frac{\partial f}{\partial w} - \frac{\partial}{\partial x_1}\frac{\partial f}{\partial w_{,x_1}} - \cdots - \frac{\partial}{\partial x_n}\frac{\partial f}{\partial w_{,x_n}} = 0. \tag{3.4}$$

For the case of multiple variables with one independent parameter, the necessary condition for the extremum of the integral

$$I = \int_{t_1}^{t_2} f(t, x_1, \ldots, x_n, \dot{x}_1, \ldots, \dot{x}_n)\, dt$$

is

$$\frac{\partial f}{\partial x_i} - \frac{d}{dt}\frac{\partial f}{\partial \dot{x}_i} = 0, \quad \text{for } i = 1, \ldots, n. \tag{3.5}$$

Similar condition can be derived for the case of several independent parameters.

Based on these observations, the equations of a beam can be derived as in the following sections.

3.2 Euler-Bernoulli Beam Equation

Several beam equations will be discussed in the following sections based on different assumptions. First, we consider the so-called *Euler-Bernoulli Beam Equation*. Since it introduces a way to derive the motion equation, as will be used in the next two sections, the details are included here as a comparison.

The beam can be modeled as in Fig. 3-1.

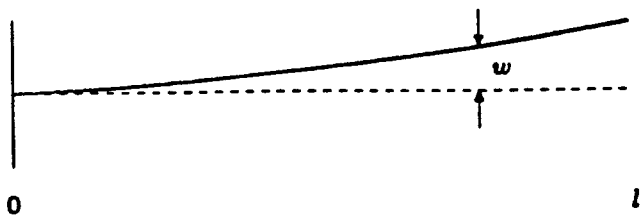


Figure 3-1 Approximate Beam Model

where $w = w(x,t)$ is the deflection from the nominal curve and l is the length of the beam. This is essentially the clamped-free case with the boundary condition

$$\begin{aligned} w(0,t) &= 0 \\ \frac{\partial w}{\partial x}(0,t) &= 0. \\ \frac{\partial^2 w}{\partial x^2}(l,t) &= 0. \\ \frac{\partial^3 w}{\partial x^3}(l,t) &= 0. \end{aligned}$$

For this case, the potential energy and kinetic energy can be found by

$$\begin{aligned} V &= \frac{1}{2}\int_0^l EI w_{,xx}{}^2\, dx \\ T &= \frac{1}{2}\int_0^l \rho \dot{w}^2\, dx \end{aligned}$$

where E is Young's modulus, I is the inertia of the cross section. Forming $L = T - V$, and applying Hamilton's principle, the integral

$$\int_{t_1}^{t_2} L dt = \int_{t_1}^{t_2}\int_0^l \frac{1}{2}\left(\rho \dot{w}^2 - EI w_{,xx}{}^2\right) dx\, dt$$

should have a stationary point. Let

$$f(\dot{w}, w_{,xx}) = \frac{1}{2}\left(\rho \dot{w}^2 - EI w_{,xx}{}^2\right),$$

since f depends explicitly on $w_{,xx}$, the previous necessary conditions derived cannot be applied. Consider

$$I = \iint_D f(\dot{w}, w_{,xx})\, dx\, dt,$$

let $W(x,t) = w(x,t) + \epsilon\eta(x,t)$, where

$$\eta(x,t) = 0, \quad \forall (x,t) \in \partial D \text{ (boundary of } D),$$

We denote

$$I(\epsilon) = \iint_D f(\dot{W}, W_{,xx})\, dx\, dt,$$

which implies

$$\frac{dI(\epsilon)}{d\epsilon} = \iint_D \left[\frac{\partial f(\dot{W}, W_{,xx})}{\partial \dot{W}} \frac{\partial \dot{W}}{\partial \epsilon} + \frac{\partial f(\dot{W}, W_{,xx})}{\partial W_{,xx}} \frac{\partial W_{,xx}}{\partial \epsilon} \right] dx\, dt. \tag{3.6}$$

Since

$$\dot{W} = \dot{w}(x,t) + \epsilon \dot{\eta}(x,t)$$

$$W_{,xx} = w_{,xx}(x,t) + \epsilon \eta_{,xx}(x,t),$$

we have

$$\frac{dI(\epsilon)}{d\epsilon} = \iint_D \left[\frac{\partial f}{\partial \dot{W}} \dot{\eta} + \frac{\partial f}{\partial W_{,xx}} \eta_{,xx} \right] dx\, dt.$$

In order to have the variation be zero, i.e. $\frac{dI(\epsilon)}{d\epsilon}|_{\epsilon=0} = 0$, we need

$$\begin{aligned} 0 = \frac{dI(\epsilon)}{d\epsilon}|_{\epsilon=0} &= \iint_D \left[\frac{\partial f}{\partial \dot{w}} \dot{\eta} + \frac{\partial f}{\partial w_{,xx}} \eta_{,xx} \right] dx\, dt \\ &= \iint_D \frac{\partial f}{\partial \dot{w}} \dot{\eta}\, dx\, dt + \iint_D \frac{\partial f}{\partial w_{,xx}} \eta_{,xx}\, dx\, dt. \end{aligned} \tag{3.7}$$

By the Green's Formula,

$$\iint_D \left(G \frac{\partial \eta}{\partial x} + F \frac{\partial \eta}{\partial y} \right) dx\, dy = -\iint_D \eta \left(\frac{\partial G}{\partial x} + \frac{\partial F}{\partial y} \right) dx\, dy + \int_{\partial D} \eta (G\, dy - F\, dx).$$

Let $G = \frac{\partial f}{\partial \dot{w}}$, $F = 0$, we have

$$\iint_D \frac{\partial f}{\partial \dot{w}} \frac{\partial \eta}{\partial t} dx\, dt = -\iint_D \eta \frac{\partial}{\partial t} \frac{\partial f}{\partial \dot{w}} dx\, dt + \int_{\partial D} \eta \frac{\partial f}{\partial \dot{w}} dx. \tag{3.8}$$

Since $\eta(x,t) = 0$ on ∂D, which implies the second term on the left hand side of Eq. 3.2 is zero, we get

$$\iint_D \frac{\partial f}{\partial \dot{w}} \frac{\partial \eta}{\partial t}\, dx\, dt = -\iint_D \eta \frac{\partial}{\partial t} \frac{\partial f}{\partial \dot{w}}\, dx\, dt. \tag{3.9}$$

By another form of Green's Formula

$$\iint_D G \frac{\partial^2 \eta}{\partial x^2} dx\, dy = \iint_D \eta \frac{\partial^2 G}{\partial x^2} dx\, dy + \int_{\partial D} \left(G \frac{\partial \eta}{\partial x} - \eta \frac{\partial G}{\partial x} \right) dy.$$

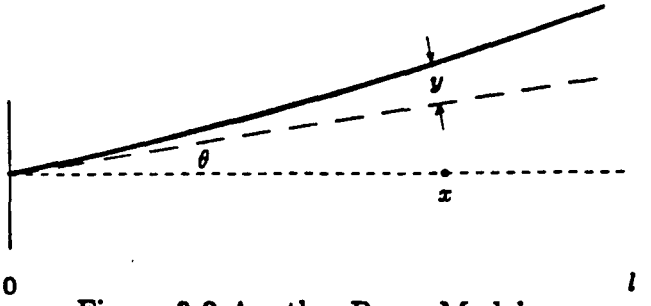


Figure 3-2 Another Beam Model

Let $G = \frac{\partial f}{\partial w_{,xx}}$, and with the boundary conditions for $\eta(x,t)$, we get

$$\iint_D \frac{\partial f}{\partial w_{,xx}} \frac{\partial^2 \eta}{\partial x^2} dx\, dt = \iint_D \eta \frac{\partial^2}{\partial x^2} \frac{\partial f}{\partial w_{,xx}} dx\, dt. \tag{3.10}$$

By plugging Eq. 3.3 and Eq. 3.4 in Eq. 3.1, we derive

$$\begin{aligned} 0 &= -\iint_D \eta \frac{\partial}{\partial t} \frac{\partial f}{\partial \dot{w}} dx\, dt + \iint_D \eta \frac{\partial^2}{\partial x^2} \frac{\partial f}{\partial w_{,xx}} dx\, dt \\ &= \iint_D \eta \left[\frac{\partial^2}{\partial x^2} \frac{\partial f}{\partial w_{,xx}} - \frac{\partial}{\partial t} \frac{\partial f}{\partial \dot{w}} \right] dx\, dt. \end{aligned} \tag{3.11}$$

By the basic lemma 3-2, we should have

$$\frac{\partial^2}{\partial x^2} \frac{\partial f}{\partial w_{,xx}} - \frac{\partial}{\partial t} \frac{\partial f}{\partial \dot{w}} = 0. \tag{3.12}$$

Since we already have

$$\frac{\partial f}{\partial w_{,xx}} = -EIw_{,xx}$$

$$\frac{\partial f}{\partial \dot{w}} = \rho \dot{w},$$

by plugging the above equations in Eq. 3.12, the equation

$$EI \frac{\partial^4 w}{\partial x^4} + \rho \frac{\partial^2 w}{\partial t^2} = 0 \tag{3.13}$$

has been derived.

3.3 Discretized Beam Equation

Next, we are going to derive another equation also illustrating the motion of the beam. Now the model of beam is shown in Fig. 3.2. The beam is mounted (hinged) on a rotational hub.

Now the large scale motion of beam is taken into account, where θ is the angle of the shadow beam, y is the bending deflection from the shadow beam, and x is the distance from the root to a certain point. With the parameters as defined before, the kinetic energy and potential energy can be now computed as

$$\begin{cases} T = \frac{1}{2}\int_0^l \rho(x\dot{\theta}+\dot{y})^2 dx \\ \quad = \frac{1}{2}\int_0^l \rho x^2 dx\ \dot{\theta}^2 + \int_0^l \rho x\dot{y} dx\ \dot{\theta} + \frac{1}{2}\int_0^l \rho \dot{y}^2 dx. \\ V = \frac{1}{2}\int_0^l EI {y_{,xx}}^2 dx + T_{\text{ext}} \end{cases} \tag{3.14}$$

where $y_{,xx} = \frac{\partial^2 y}{\partial x^2}$. Since now the beam is rotating through an angle θ, there should be some external torque, i.e. T_{ext} and assume that $\delta T_{\text{ext}} = -\tau\delta\theta$.

Here the discretization method will be used in representing y, i.e. assume

$$y = \sum_{i=1}^{n} q_i(t)\phi_i(x).$$

It is essentially a finite approximation of n modes. Thus,

$$\dot{y} = \sum_{i=1}^{n} \dot{q}_i(t)\phi_i(x)$$
$$\frac{\partial^2 y}{\partial x^2} = \sum_{i=1}^{n} q_i(t)\phi_{i,xx}(x),$$

where $\phi_{i,xx} = \frac{\partial^2 \phi_i}{\partial x^2}$. By plugging $\dot{y}$ and $y_{,xx}$ into the equation (3.14),

$$\begin{cases} T = \frac{1}{2}\int_0^l \rho x^2 dx\ \dot{\theta}^2 + \int_0^l \rho x \sum_i \dot{q}_i(t)\phi_i(x)\, dx\ \dot{\theta} + \frac{1}{2}\int_0^l \rho \left(\sum_i \dot{q}_i(t)\phi_i(x)\right)^2 dx \\ V = \frac{1}{2}\int_0^l EI \left(\sum_{i=1}^{n} q_i(t)\phi_{i,xx}(x)\right)^2 dx + T_{\text{ext}}. \end{cases} \tag{3.15}$$

Letting $L = T - V$, by Hamilton's principle,

$$\delta \int_{t_1}^{t_2} L dt = 0.$$

Since $L = L(t, \theta, \dot{\theta}, q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n)$, this is case of one parameter, multiple variables case. Thus the formula (3.5) can be used, i.e.

$$\begin{cases} \dfrac{\partial L}{\partial \theta} - \dfrac{d}{dt}\dfrac{\partial L}{\partial \dot{\theta}} = 0 \\ \dfrac{\partial L}{\partial q_i} - \dfrac{d}{dt}\dfrac{\partial L}{\partial \dot{q}_i} = 0, \quad \text{for} i = 1, \ldots, n. \end{cases} \tag{3.16}$$

Since

$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \tau \\ \frac{\partial L}{\partial \dot{\theta}} &= \int_0^l \rho x^2 dx \dot{\theta} + \int_0^l \rho x \sum_i \dot{q}_i \phi_i dx \\ \frac{\partial L}{\partial q_j} &= -\int_0^l EI(\sum_i q_i \phi_{i,xx})\ \phi_{j,xx} dx \\ \frac{\partial L}{\partial \dot{q}_j} &= \int_0^l \rho x \phi_j dx \dot{\theta} + \int_0^l \rho (\sum_i \dot{q}_i \phi_i) \phi_j dx, \end{aligned}$$

by plugging them in Eq. (3.16), we get

$$\begin{cases} \displaystyle\int_0^l \rho x^2 dx \ddot{\theta} + \sum_i \int_0^l \rho x \phi_i dx \ddot{q}_i = \tau \\ \displaystyle\sum_i \left(\int_0^l EI \phi_{i,xx} \phi_{j,xx} dx q_i + \int_0^l \rho \phi_i \phi_j dx \ddot{q}_i \right) = -\int_0^l \rho x \phi_j dx \ddot{\theta}, \quad j = 1, \ldots, n. \end{cases} \tag{3.17}$$

If Eq. 3.17 is put into the state-space form, e.g. $m\ddot{q} + kq = b\tau$, then it is exactly the same equation as derived in [16] .

3.4 Geometrically Exact Beam Equation

Another model, the so-called *geometrically exact model* for the plane motion of the beam will be derived as follows. It takes account of the shear deformation as well as the flexure deformation in the motion. Now the motion of the beam can be drawn as in Fig. 3-3, where $\{\hat{e}_I, I = 1, 2, 3\}$ is the reference coordinates with $\hat{e}_3 = \hat{e}_1 \times \hat{e}_2$, $\{\hat{t}_I, I = 1, 2, 3\}$ is the deformed coordinates with $\hat{t}_3 = \hat{t}_1 \times \hat{t}_2$, $\hat{X}$ is a point on the beam in the reference configuration and $\hat{x}$ is the point where the $\hat{X}$ has been moved to.

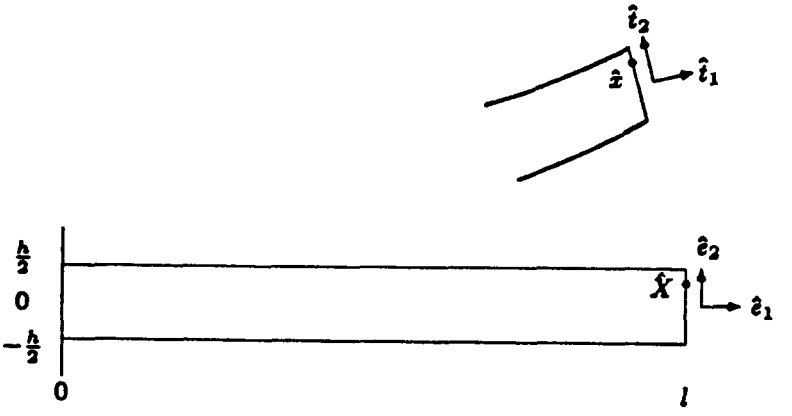


Figure 3-3 Exact Beam Model

$\hat{X}$ and $\hat{x}$ can be represented in the form as, with different coordinates,

$$\begin{aligned}\hat{X} &= X_1\hat{e}_1 + X_2\hat{e}_2 \\ \hat{x} &= \hat{\phi}_0(X_1,t) + X_2\hat{t}_2 \\ &= [X_1 + u_1(X_1,t)]\,\hat{e}_1 + [u_2(X_1,t)]\,\hat{e}_2 + X_2\hat{t}_2.\end{aligned} \tag{3.18}$$

where $\hat{\phi}_0(X_1,t)$ is the line of centroid of the beam, and the axis transformation is

$$\begin{cases}\hat{t}_1 = \cos\theta\hat{e}_1 + \sin\theta\hat{e}_2 \\ \hat{t}_2 = -\sin\theta\hat{e}_1 + \cos\theta\hat{e}_2 \\ \hat{t}_3 = \hat{e}_3.\end{cases} \tag{3.19}$$

and

$$\frac{\partial\hat{t}_1}{\partial X_1} = \frac{\partial\theta}{\partial X_1}\hat{t}_2, \qquad \frac{\partial\hat{t}_1}{\partial t} = \dot{\theta}\hat{t}_2,$$
$$\frac{\partial\hat{t}_2}{\partial X_1} = -\frac{\partial\theta}{\partial X_1}\hat{t}_1, \qquad \frac{\partial\hat{t}_2}{\partial t} = -\dot{\theta}\hat{t}_1.$$

So the derivative of $\hat{x}$ can be found as

$$\begin{aligned}\dot{\hat{x}} &= \dot{\hat{\phi}}_0 + X_2\dot{\hat{t}}_2 \\ &= (\dot{u}_1 - \dot{\theta}X_2\cos\theta)\hat{e}_1 + (\dot{u}_2 - \dot{\theta}X_2\sin\theta)\hat{e}_2.\end{aligned}$$

and

$$\begin{aligned}\|\dot{\hat{x}}\|^2 &= \dot{\hat{x}}\cdot\dot{\hat{x}} \\ &= (\dot{u}_1^2 + \dot{u}_2^2) + \dot{\theta}^2X_2^2 - 2\dot{\theta}X_2(\dot{u}_1\cos\theta + \dot{u}_2\sin\theta).\end{aligned}$$

The kinetic energy of the system can be then computed as

$$
\begin{aligned}
T &= \frac{1}{2}\int_0^l \rho \|\dot{\hat{x}}\|^2 dX_1 \\
&= \frac{1}{2}\int_0^l \int_{\frac{-h}{2}}^{\frac{h}{2}} \rho \left[(\dot{u}_1^2 + \dot{u}_2^2) + \dot{\theta}^2 X_2^2 - 2\dot{\theta} X_2 (\dot{u}_1 \cos\theta + \dot{u}_2 \sin\theta)\right] dX_2 dX_1 \\
&= \frac{1}{2}\int_0^l \Big[\Big(\int_{\frac{-h}{2}}^{\frac{h}{2}} \rho dX_2\Big)(\dot{u}_1^2 + \dot{u}_2^2) + \Big(\int_{\frac{-h}{2}}^{\frac{h}{2}} \rho X_2^2 dX_2\Big)\dot{\theta}^2 \\
&\qquad - 2\Big(\int_{\frac{-h}{2}}^{\frac{h}{2}} \rho X_2 dX_2\Big)\dot{\theta}(\dot{u}_1 \cos\theta + \dot{u}_2 \sin\theta)\Big] dX_1
\end{aligned}
$$

Assume that the beam is symmetric, the integral $\int_{\frac{-h}{2}}^{\frac{h}{2}} \rho X_2 dX_2 = 0$. Let

$$
\begin{aligned}
A_\rho &= \int_{\frac{-h}{2}}^{\frac{h}{2}} \rho dX_2, \quad \text{and} \\
I_\rho &= \int_{\frac{-h}{2}}^{\frac{h}{2}} \rho X_2^2 dX_2.
\end{aligned}
$$

Then, Eq. 3.13 is reduced to

$$
T = \frac{1}{2}\int_0^l \left[A_\rho(\dot{u}_1^2 + \dot{u}_2^2) + I_\rho \dot{\theta}^2\right] dX_1.
$$

Next, the potential energy will be constructed. First, the strain field is introduced. When the beam undergoes large strains, the strain field can be defined as

$$
\begin{aligned}
\hat{\gamma} &\triangleq \frac{\partial \hat{\phi}_0}{\partial X_1} - \hat{t}_1 \\
&= (1 + u_{1,x} - \cos\theta)\hat{e}_1 + (u_{2,x} - \sin\theta)\hat{e}_2.
\end{aligned} \tag{3.20}
$$

($u_{1,x}$ really means that $\frac{\partial u_1}{\partial X_1}$, for the reason of simplicity, the simpler notation will be used here as well as in the subsequent discussions.) Then, relative to the coordinate system $\{\hat{t}_I\}$, from Eq. 3.16, this field can be written as

$$
\begin{aligned}
\hat{\gamma} &= (1 + u_{1,x} - \cos\theta)(\cos\theta\hat{t}_1 - \sin\theta\hat{t}_2) + (u_{2,x} - \sin\theta)(\sin\theta\hat{t}_1 + \cos\hat{t}_2) \\
&= (\cos\theta + u_{1,x}\cos\theta + u_{2,x}\sin\theta - 1)\hat{t}_1 + (-\sin\theta - u_{1,x}\sin\theta + u_{2,x}\cos\theta)\hat{t}_2.
\end{aligned}
$$

Now the axial strain Γ_1 and shearing strain Γ_2 are defined as, respectively,

$$
\begin{cases}
\Gamma_1 = \cos\theta + u_{1,x}\cos\theta + u_{2,x}\sin\theta - 1 \\
\Gamma_2 = -\sin\theta - u_{1,x}\sin\theta + u_{2,x}\cos\theta.
\end{cases} \tag{3.21}
$$

Let EA be the *axial stiffness*, GA_s be the *shear stiffness*, and EI be the *flexural stiffness*, then the potential energy can be written as

$$V = \frac{1}{2}\int_0^l (EA\Gamma_1^2 + GA_s\Gamma_2^2 + EI\theta_{,x}^2)\, dX_1 - \Pi_{ext} - T_{ext}(t)\theta(0,t) \tag{3.22}$$

where Π_{ext} is the potential energy of the external loading acting on the beam and $T_{ext}(t)\hat{e}_3$ is the applied torque at the axis of rotation $\hat{e}_3$ of the beam. Now we assume that there is no external loading, i.e. $\Pi_{ext} = 0$. Then, the lagrangian L can be found as

$$\begin{aligned} L &= T - V \\ &= \int_0^l \left[\frac{1}{2}A_\rho(\dot{u}_1^2 + \dot{u}_2^2) + \frac{1}{2}I_\rho\dot{\theta}^2 - \frac{1}{2}(EA\Gamma_1^2 + GA_s\Gamma_2^2 + EI\theta_{,x}^2) + \frac{T_{ext}}{l}\theta\right] dX_1. \end{aligned} \tag{3.23}$$

Hamilton's principle requires that $\int_{t_1}^{t_2} L\, dt$ to be stationary. Let

$$\begin{aligned} &f(X_1, t, u_1, u_2, \theta, \dot{u}_1, \dot{u}_2, \dot{\theta}, u_{1,x}, u_{1,x}, \theta_{,x}) \\ &= \frac{1}{2}A_\rho(\dot{u}_1^2 + \dot{u}_2^2) + \frac{1}{2}I_\rho\dot{\theta}^2 - \frac{1}{2}(EA\Gamma_1^2 + GA_s\Gamma_2^2 + EI\theta_{,x}^2) + \frac{T_{ext}}{l}\theta. \end{aligned}$$

Then, it is the case of three dependent variables with two independent parameters. By the formula from the calculus of variations, analogous to Eq. 3.3, the equations are

$$\begin{cases} \dfrac{\partial f}{\partial u_1} - \dfrac{\partial}{\partial X_1}\dfrac{\partial f}{\partial u_{1,x}} - \dfrac{\partial}{\partial t}\dfrac{\partial f}{\partial \dot{u}_1} = 0 \\ \dfrac{\partial f}{\partial u_2} - \dfrac{\partial}{\partial X_1}\dfrac{\partial f}{\partial u_{2,x}} - \dfrac{\partial}{\partial t}\dfrac{\partial f}{\partial \dot{u}_2} = 0 \\ \dfrac{\partial f}{\partial \theta} - \dfrac{\partial}{\partial X_1}\dfrac{\partial f}{\partial \theta_{,x}} - \dfrac{\partial}{\partial t}\dfrac{\partial f}{\partial \dot{\theta}} = 0. \end{cases} \tag{3.24}$$

The only thing left is to find the partial differentiations and then plug into Eq. 3.24. We have

$$\begin{cases} \dfrac{\partial f}{\partial u_1} = 0 \\ \dfrac{\partial f}{\partial u_{1,x}} = -EA\Gamma_1\cos\theta + GA_s\Gamma_2\sin\theta \triangleq n_1 \\ \dfrac{\partial f}{\partial \dot{u}_1} = A_\rho\dot{u}_1. \end{cases}$$

thus, by the first equation in Eq. 3.24,

$$A_\rho\ddot{u}_1 + n_{1,x} = 0. \tag{3.25}$$

From

$$\begin{cases} \dfrac{\partial f}{\partial u_2} = 0 \\ \dfrac{\partial f}{\partial u_{2,x}} = -EA\Gamma_1 \sin\theta - GA_s\Gamma_2 \cos\theta \triangleq n_2 \\ \dfrac{\partial f}{\partial \dot{u}_2} = A_\rho \ddot{u}_2, \end{cases}$$

we get

$$A_\rho \ddot{u}_2 + n_{2,x} = 0. \tag{3.26}$$

From

$$\begin{cases} \dfrac{\partial f}{\partial \theta} = EA\Gamma_1 \sin\theta + GA_s\Gamma_2 \cos\theta + u_{1,x}(EA\Gamma_1 \sin\theta + GA_s\Gamma_2 \cos\theta) \\ \qquad\qquad + u_{2,x}(-EA\Gamma_1 \cos\theta + GA_s\Gamma_2 \sin\theta) + \dfrac{T_{\text{ext}}}{l} \\ \qquad = -n_2 - u_{1,x}n_2 + u_{2,x}n_1 + \dfrac{T_{\text{ext}}}{l} \\ \dfrac{\partial f}{\partial \theta_{,x}} = -EI\theta_{,x} \\ \dfrac{\partial f}{\partial \dot{\theta}} = I_\rho \dot{\theta}, \end{cases}$$

the third equation can be obtained as

$$I_\rho \ddot{\theta} + (1 + u_{1,x})n_2 - u_{2,x}n_1 - EI\theta_{,xx} = \frac{T_{\text{ext}}}{l}. \tag{3.27}$$

From Eq. 3.25, 3.26, 3.27, the governing equation can be then written in the matrix form,

$$\begin{bmatrix} A_\rho & 0 & 0 \\ 0 & A_\rho & 0 \\ 0 & 0 & I_\rho \end{bmatrix} \begin{bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} n_{1,x} \\ n_{2,x} \\ (1 + u_{1,x})n_2 - u_{2,x}n_1 - EI\theta_{,xx} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{T_{\text{ext}}}{l} \end{bmatrix}. \tag{3.28}$$

This is the geometrically exact beam equation for the planer motion of a flexible arm. If the assumption of $\Pi_{\text{ext}} = 0$ in Eq. 3.22 is changed to

$$\Pi_{\text{ext}} = \int_0^l (\hat{\bar{m}} \cdot \theta \hat{e}_3 + \hat{\bar{n}} \cdot \hat{\phi}_0)\, dX_1$$

with

$$\hat{\bar{m}} = \bar{m}(X_1, t)\hat{e}_3$$

$$\hat{\bar{n}} = \bar{n}_1(X_1, t)\hat{e}_1 + \bar{n}_2(X_1, t)\hat{e}_2,$$

and $T_{ext} = 0$ is assumed, then, by the same derivation method, the beam equation is exactly the same with in [17] .

Although this beam equation is quite realistic, for the reason that it is too complicated, it is not used in the following chapters. The beam-hub model used later is essentially borrowed from [11] , and is experimentally determined.

CHAPTER IV

CONSOLE Design

4.1 Introduction

Optimization-based design is an effective means to solve design problems with constraints. Although there is a number of sophisticated methods in the field of optimal control to fulfill this purpose, few among them take the advantages of recent powerful optimization algorithms and numerical CAD package in doing the design. DELIGHT.Marylin[13] is such an optimization tool for designing linear time invariant control systems. It is however cumbersome to use and to maintain. Another optimization package called CONSOLE has been recently developed. It mainly consists of two programs: CONVERT and SOLVE, and can be connected with any simulators designed by the user, e.g. *MaryLin* (a simulator for linear time-invariant control systems.) With the help of CONSOLE , the designer can automatically adjust certain controller parameters to meet some specific requirements without getting involved in the details of optimal control theory.

The details of how to use CONSOLE and how CONSOLE works can be found in the CONSOLE manual [10]. The next section will give an example of how to design a controller for the flexible arm by using CONSOLE with the simulator *MaryLin* .

4.2 Designing a Controller for the Flexible Arm

The flexible arm described in Chapter 2 has been identified as a linear time-invariant system[11] and its transfer function from hub position to tip acceleration

is modeled as

$$H(s) = \frac{0.125s[\frac{s^2}{(2\pi \times 9.5)^2} + \frac{2.1s}{2\pi \times 9.5} + 1]}{(\frac{s}{2\pi \times 1.16} + 1)[\frac{s^2}{(2\pi \times 9)^2} + \frac{2(0.11)s}{2\pi \times 9} + 1][\frac{s^2}{(2\pi \times 20.6)^2} + \frac{2(0.02)s}{2\pi \times 20.6} + 1]}$$

In order to reach the states by the three sensors, position encoder, tachometer, accelerometer, the above transfer function can be decomposed as

$$H(s) = \underbrace{\frac{2.454}{s(\frac{s}{7.3} + 1)}}_{motor} \underbrace{\frac{0.134[\frac{s^2}{(19\pi)^2} + \frac{2.1s}{19\pi} + 1]}{[\frac{s^2}{(18\pi)^2} + \frac{0.22s}{18\pi} + 1][\frac{s^2}{(41.2\pi)^2} + \frac{0.04s}{41.2\pi} + 1]}}_{beam} \underbrace{\frac{0.383s^2}{1}}_{acceleration}$$

The state space representation of the system is thus the following:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} -17.618 & 1 & 0 & 0 & 0 & 0 \\ -20015.2 & 0 & 1 & 0 & 15035.99 & 0 \\ -224975.8 & 0 & 0 & 1 & 1795004.1 & 0 \\ -53572128.7 & 0 & 0 & 0 & 53572128.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -7.29 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 17.96 \end{bmatrix} u$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -999.6 & -0.89470.0507 & 0 & 762.73 & 0 & \end{bmatrix} x$$

where y_1 is the tip position, y_2 is the tip acceleration.

The state feedback controller is chosen to attain the requirements, i.e. $u = -Kx + v$, where v is the reference input. Thus, there are six design parameters in this problem. Listing 4-1 shows the problem description file which will be read by CONVERT.

Listing 4-1 Problem Description File

```
PI = 3.141592653
R1 = 1/(9*2*PI)
R2 = 1/(20.6*2*PI)
R3 = 1/(9.5*2*PI)

design_parameter k1 init=8.234939958e+00 vari=1.43e0
design_parameter k2 init=-1.58345004e-01 vari=9e-2
design_parameter k3 init=-1.579990251e-03 vari=8.4e-4
design_parameter k4 init=2.387055555e-5 vari=1.7e-5
```

```
design_parameter k5 init=4.089939087e1 vari=4.34e1
design_parameter k6 init=5.853e-1 vari=2e-1

functional_objective "over-shoot"
   for t from 0 to 3 by .01
   minimize {
      double Ytr();
      return Ytr("1",t);
      }
   good_curve = {
       if( t <= 1. ) return 1.08;
       else          return 1.02;
       }
   bad_curve = {
       if( t <= 1. ) return 1.15;
       else          return 1.05;
       }
functional_objective "settling"
   for t from 0.6 to 3 by .01
   maximize {
      double Ytr();
      return Ytr("1",t);
      }
   good_curve = {
       if( t <= 0.8 ) return 0.9;
       else           return 0.99;
       }
   bad_curve = {
       if( t <= 0.8 ) return 0.85;
       else           return 0.95;
       }

objective "steadystate"
   minimize {
      double Ytr();
      return fabs( Ytr("1",5.0)-1 );
      }
   good_value = 0
   bad_value =  1e-6

functional_constraint "max-accel" hard
   for t from 0 to 3 by .01
   {
      double Ytr();
      return Ytr("2",t);
   }
   <=
   good_curve = { return 10; }
   bad_curve = { return 11; }
functional_constraint "max-accel" hard
   for t from 0 to 3 by .01
   {
      double Ytr();
      return Ytr("2",t);
   }
   >=
   good_curve = { return -10; }
   bad_curve = { return -11; }
functional_constraint "control" hard
   for t from 0 to 3 by .01
   {
```

```
    import k1 k2 k3 k4 k5 k6
    double Ytr();
    return fabs( 1-(k1*Ytr("3",t)+k2*Ytr("4",t)+k3*Ytr("5",t)+k4*Ytr("6",t)
          +k5*Ytr("7",t)+k6*Ytr("8",t) ));
} <=
good_curve = {
    return 4;
    }
bad_curve = {
    return 5;
    }
```

The desired position profile is shown as in Fig. 4-1, and is set by the first two objectives.

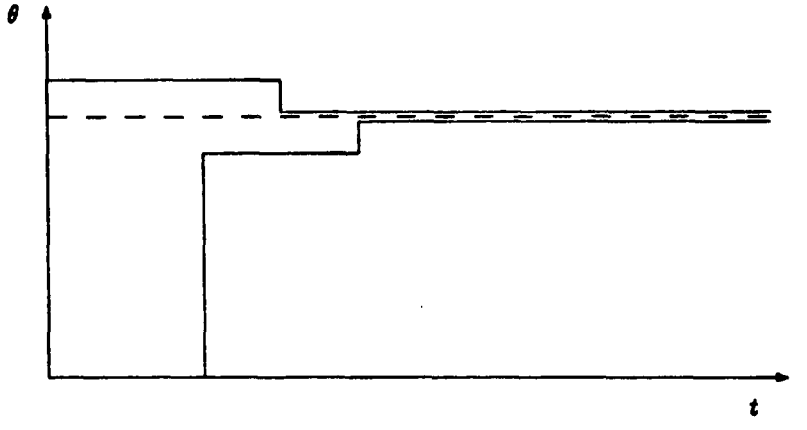


Figure 4-1 Desired Position Profile

Since the control signal (voltage across the motor) can not be too large, the constraint for it is set by the functional_constraint named "control". The constraint for the tip acceleration is set by the other two functional_constraints to limit the tip acceleration in certain range.

Listing 4-2 gives the system description which will be read by *MaryLin* and linked with SOLVE. It is a straightforward translation from the state representation of the system as in Eq. 4.1. The feedback gains were included in the A matrix. The reference input now is set by Ut.

Listing 4-2 System Description File

```
PI = 3.1415926
G = 0.125
R1 = 9*2*PI
R2 = 20.6*2*PI
R3 = 9.5*2*PI
R4 = G / (0.383 * 2.124)
```

```
A1 = 0.04*R2+0.22*R1
A2 = R1*R1+0.0088*R2*R1+R2*R2
A3 = 0.22*R2*R2*R1+0.04*R2*R1*R1
A4 = R2*R2*R1*R1
B2 = R1*R2*R1*R2/(R3*R3)
B3 = 2*B2*R3
B4 = B2*R3*R3
C1 = (A1*A1-A2)*0.383*R4
C2 = -A1*0.383*R4
C3 = 0.383*R4
C4 = B2*0.383*R4
b6 = 2*PI*1.16*2.124

system_size Ninputs=1 Nstates=6 Noutputs=8
readmatrix A
-A1          1          0          0          0          0
-A2          0          1          0          B2         0
-A3          0          0          1          B3         0
-A4          0          0          0          B4         0
0            0          0          0          0          1
-b6*k1     -b6*k2     -b6*k3     -b6*k4     -b6*k5     -7.29-b6*k6

readmatrix  B
0
0
0
0
0
b6

readmatrix C
1          0          0          0          0          0
C1         C2         C3         0          C4         0
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
0          0          0          0          0          1

readmatrix Ut
1
```

The CONSOLE tandem works as follows: CONVERT reads the problem description file and SOLVE then calls the simulator *MaryLin* which reads the system description file to optimize the system performance subject to the changing of design parameters.

The initial guess of the design parameters is chosen in the same way as in [11]. By interacting with SOLVE for several iterations, the final values of the design parameters were obtained as given in Listing 4-3. The *pcomb* (performance comb) output is also given there. It shows how close each of the constraints or objectives is to their corresponding good and bad value.

Listing 4-3 Designed Value of Design Parameters and *Pcomb* Output

```
Name        Value        Variation  wrt 0   Prev    Iter=30
k1          8.23494e+00  1.4e+00
k2         -1.58345e-01  9.0e-02
k3         -1.57999e-03  8.4e-04
k4          2.38706e-05  1.7e-05
k5          4.08994e+01  4.3e+01
k6          5.85300e-01  2.0e-01

Pcomb (Iter=  30) (Phase 2) (eps= 1.000e+00) (MAX_COST_SOFT= 0.9341)

SPECIFICATION    PRESENT   GOOD              G          B            BAD
O1  steadystat  6.72e-05  0.00e+00 ==================*   |            1.00e-04
FO1 over-shoot  1.09e+00  1.01e+00 ====================*|            1.10e+00
FO2 settling    9.91e-01  9.90e-01           *==========|=========  9.50e-01
FC1 max-accel   2.71e+00  1.00e+01 <--        |          |           1.10e+01
FC2 max-accel  -6.71e-01 -1.00e+01 <-------------------------------- -1.10e+01
FC3 control     1.03e+00  4.00e+00 <--        |          |           5.00e+00
```

The plots of tip position and control signal are shown in Fig. 4-2 and Fig. 4-3, respectively. With this design, the control law can be then implemented in the real-time system. The details of the implementation will be described in Chapter 7.

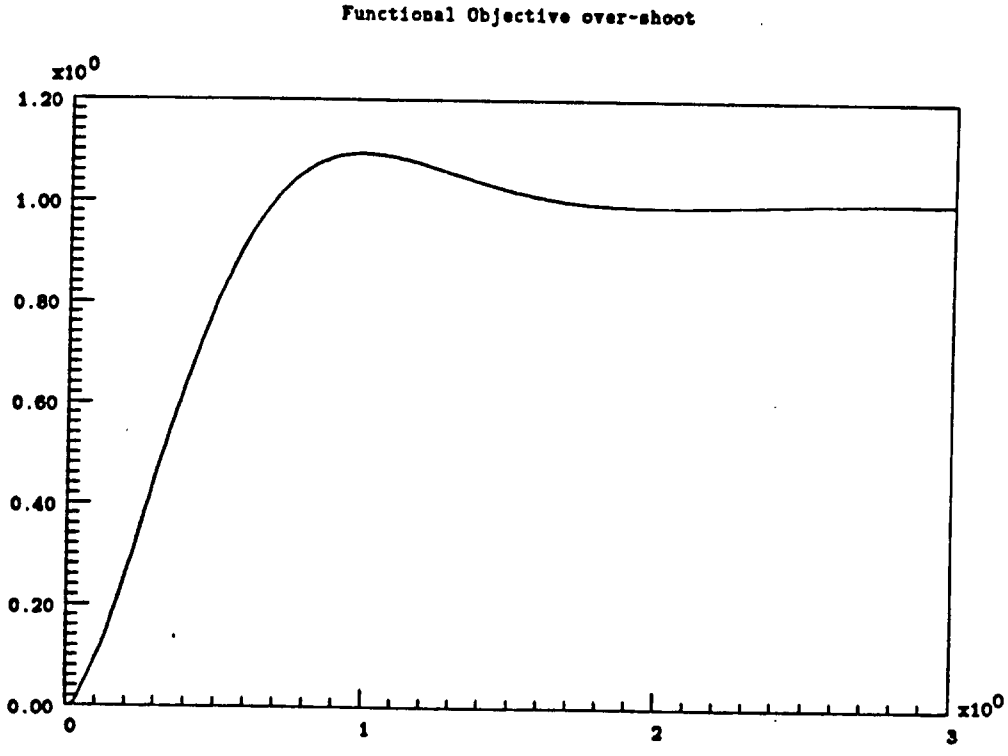


Figure 4-2 Plot of Tip Position

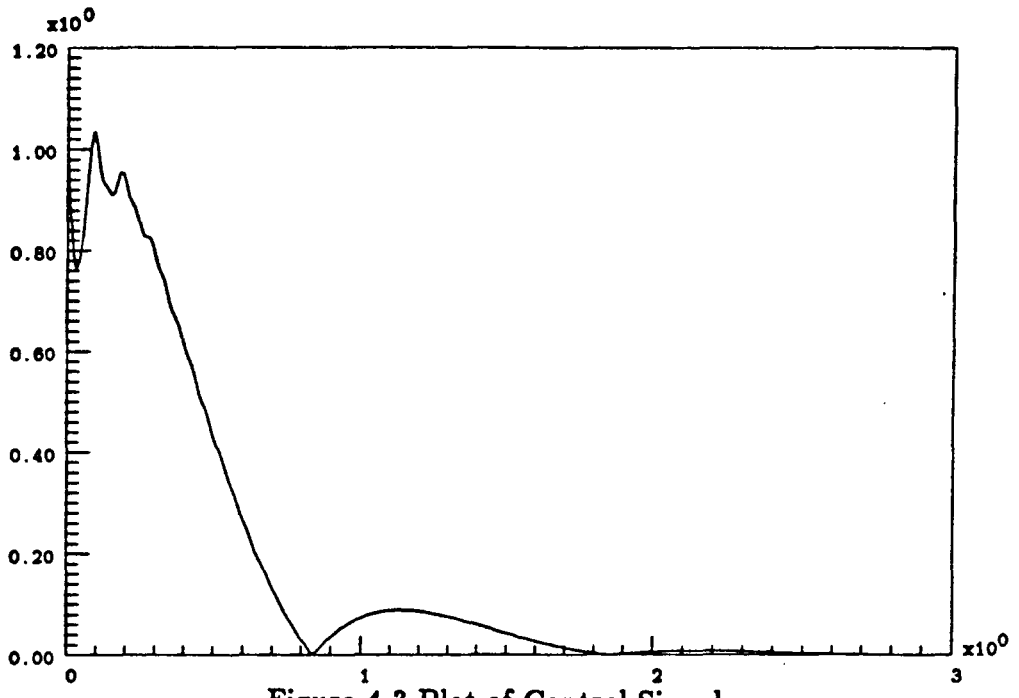


Figure 4-3 Plot of Control Signal

CHAPTER V

Integral Control

5.1 Introduction

In order to compensate steady state errors, we use a scheme based on integral control. The system is augmented with an artificial state which will be used in the integral feedback. This chapter will investigate some properties of the augmented system and how to use this idea in the control of a motor.

5.2 The Augmented System (SISO Case)

The general SISO(single-input-single-output) system can be modeled as

$$\begin{cases} \dot{x} = Ax + bu \\ y = cx + du \end{cases} \tag{5.1}$$

where

$$A : n \times n,\ b : n \times 1,\ c : 1 \times n,\ d : 1 \times 1.$$

The idea is to use the integral of the output as part of the feedback. The control law may be written as

$$u = -Kx - K_I \int y + v \tag{5.2}$$

where v is the reference input.

The integral term is accounted for by a new state z. Append to Eq. 5.1 the equation

$$\dot{z} = pz + my. \tag{5.3}$$

Then for $p = 0$, the new state z is exactly $m \int y$. Therefore, the control law in Eq. 5.2 is exactly

$$u = -Kx - \frac{K_I}{m} z + v \tag{5.4}$$

The integral control problem is thus transformed into a state-feedback control problem with an augmented state z.

The new system may be written as

$$\dot{x} = Ax + bu$$

$$\dot{z} = pz + my$$

$$y = cx + du$$

where p, m are scalars. By plugging y into $\dot{z}$,

$$\dot{z} = pz + mcx + mdu,$$

and the augmented system can be represented as

$$\begin{cases} \begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} A & 0 \\ mc & p \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} b \\ md \end{bmatrix} u \\ y = [c \;\; 0] \begin{bmatrix} x \\ z \end{bmatrix} + du \end{cases} \tag{5.5}$$

Following a discussion of the properties of the augmented system, we will present the controller design using state feedback for this new system.

5.3 Some Properties of The Augmented System (SISO Case)

This section will investigate some properties of the new system (5.5). Obviously, if the original system (5.1) is stable, then the augmented system (5.5) is stable when $p < 0$. For the controllability and observability of the augmented system, the following two propositions give the necessary and sufficient condition.

Proposition 5.1

Assume the original system (5.1) is controllable, and let its transfer function $h(s)$ to be $\frac{q(s)}{a(s)}$. Then the augmented system (5.5), with $m \neq 0$, is controllable if and only if p is not a root of $q(s)$.

Proof

By the PBH rank test for controllability[9], first form the matrix

$$\bar{Q} = [\, sI - \bar{A} \quad \bar{b} \,]$$

$$= \begin{bmatrix} sI - A & 0 & b \\ -mc & s - p & md \end{bmatrix}$$

The system (5.5) is controllable if and only if $\bar{Q}$ is of full rank for all s.

For $s \neq p$, the last row is independent of the other rows, since the original system is controllable, thus $\bar{Q}$ is of full rank. For the case of $s = p$,

$$\begin{aligned}
& \bar{Q} \text{ is of full rank} \\
\Longleftrightarrow \quad & \begin{bmatrix} pI - A & b \\ -mc & md \end{bmatrix} \text{ is nonsingular,} \\
\Longleftrightarrow \quad & \det \begin{bmatrix} pI - A & b \\ -mc & md \end{bmatrix} \neq 0, \\
\Longleftrightarrow \quad & \det(pI - A)\{mc(pI - A)^{-1}b + md\} \neq 0, \\
\Longleftrightarrow \quad & ma(p)h(p) \neq 0, \\
\Longleftrightarrow \quad & mq(p) \neq 0, \\
\Longleftrightarrow \quad & p \text{ is not a root of } q(s).
\end{aligned}$$

QED ∎

As for the property of observability of the augmented system, if the matrix

$$\bar{R} = \begin{bmatrix} sI - \bar{A} \\ \bar{c} \end{bmatrix} = \begin{bmatrix} sI - A & 0 \\ -mc & s - p \\ c & 0 \end{bmatrix}$$

is formed, then for $s = p$, it will never be of full rank. However if we revise the augmented system to be observed as

$$y = \begin{bmatrix} c & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + du, \tag{5.6}$$

then since the state z is artificial, it can always be observed. It is easy to get the new output from the old one. With this new revised system, the following proposition gives the condition for preserving observability.

Proposition 5.2

Assume the original system (5.1) is observable. The revised augmented system with output (5.6) is observable if and only if $p - m$ is not an eigenvalue of matrix A.

Proof

As before, by the PBH rank test for the observability with

$$\bar{R} = \begin{bmatrix} sI - A & 0 \\ -mc & s - p \\ c & 1 \end{bmatrix}$$

For $s = p$,

$$\bar{R} = \begin{bmatrix} sI - A & 0 \\ -mc & 0 \\ c & 1 \end{bmatrix}$$

Obviously, $\bar{R}$ is of full rank when the original system is observable. For $s \neq p$, by the elementary row operations, $\bar{R}$ is equivalent to

$$\bar{R} \sim \begin{bmatrix} sI - A & 0 \\ (-m + p - s)c & 0 \\ c & 1 \end{bmatrix}$$

Thus, for $\bar{R}$ to be of full rank,

$$\begin{bmatrix} sI - A \\ (-m + p - s)c \end{bmatrix} \quad \text{must be of full rank for all } s \neq p$$

For $-m + p - s \neq 0$, it is directly implied. For $-m + p - s = 0$, we should consider the matrix

$$\begin{bmatrix} (p - m)I - A \\ 0 \end{bmatrix} \quad \text{to be of full rank}$$

$\Longleftrightarrow$ $[(p - m)I - A]$ must be nonsingular,

$\Longleftrightarrow$ $\det[(p - m)I - A] \neq 0$,

$\Longleftrightarrow$ $a(p - m) \neq 0$,

$\Longleftrightarrow$ $p - m$ is not a root of $a(s)$.

$\Longleftrightarrow$ i.e. $p - m$ is not an eigenvalue of matrix A.

QED ∎

5.4 Designing the Controller with CONSOLE

The feedback gains K, K_I, as well as the augmented state parameters m, p, can be designed by using CONSOLE . Simply declare the state parameter p, m and the feedback gain K, K_I as the design parameters, with the simulator *MaryLin* . The optimization process in CONSOLE is used to get a best design in some sense. The conditions for stability, controllability, observability can be set as constraints. Next, a simple example will be given to illustrating this method. The application to the flexible arm control will be described later.

Example

A motor system can be modeled as

$$\begin{aligned} J\dot{\omega} &= \frac{KK_t}{R_a}U - \frac{K_t}{R_aK_b}\omega \\ \dot{\theta} &= \omega. \end{aligned} \tag{5.7}$$

Let $x_1 = \theta$, and $x_2 = \omega$, the state-space representation of the whole system may be written as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{K_t}{JR_aK_b} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{KK_t}{JR_a} \end{bmatrix} U$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

By adding the augmented state equation $\dot{z} = -pz + my$ (note that in order to preserve the positiveness of p, a minus sign included), the augmented system is

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{K_t}{JR_aK_b} & 0 \\ m & 0 & -p \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{KK_t}{JR_a} \\ 0 \end{bmatrix} U$$

$$y = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ z \end{bmatrix}$$

The transfer function of the original system is

$$H(s) = \frac{\frac{KK_t}{JR_a}}{s\left(s + \frac{K_t}{JR_aK_b}\right)}$$

Thus, by the Proposition 5-1 and 5-2, $-p - m$ cannot be 0 or $-\frac{K_t}{JR_aK_b}$ and there is no constraint for p.

The controller is chosen as $U = -K_1x_1 - K_2x_2 - K_3z + V$. By letting m, p, K_1, K_2, K_3 be the design parameters, CONSOLE can be used to design the system. Listing 5- 1 shows the input file to CONSOLE .

Listing 5-1 Input File to CONSOLE for Integral Control

```
/***   Integral Control Problem: problem description file   ***/
Kt = 21.62
Kb = 2.443
Ra = 33.6
J = 0.1
```

```
design_parameter K1  init=8.38833 vari=3
design_parameter K2  init=-1.61255e-1 vari=0.09
design_parameter K3  init=-1.37716e1 vari=6
design_parameter m  init=1.33839e1 vari=5
design_parameter p  init=2.49271e1 min=0.01 vari=10
/***   Meet the engineering specification  ***/
functional_objective "over"
   for t from 0 to 2 by 0.005
   minimize {   double Ytr();
                return Ytr(1,t); }
   good_curve = {   if( t <= 0.5 ) return 1.1
                    else          return 1.01; }
   bad_curve = {    if( t <= 0.5 ) return 1.5;
                    else          return 1.05; }
functional_objective "under"
   for t from 0.4 to 2 by 0.005
   maximize {   double Ytr();
                return Ytr(1,t); }
   good_curve = {   if( t <= 0.6 ) return 0.90;
                    else           return 0.99; }
   bad_curve = {    if( t <= 0.6 ) return 0.85;
                    else           return 0.95;  }
constraint "observ" hard
   {   import m p Kt Ra Kb J
       return fabs( (-p-m)*(-p-m)+Kt*(-p-m)/(J*Ra*Kb) );
   }   >=   good_value=1
            bad_value =0.0001

#####   System Description File for integral control   #####
K = 6
Kt = 21.62
Kb = 2.443
Ra = 33.6
J = 0.1
G = K * Kt / (J * Ra)
system_size Ninputs=1 Nstates=3 Noutputs=1
readmatrix A
0           1                          0
-G*K1      -Kt/(J*Ra*Kb)-G*K2         -G*K3
m           0                         -p
readmatrix  B
0
G
0
readmatrix C
1    0    0
readmatrix Ut
1
```

The constraint "observ" is set for satisfying the condition of $-p-m$, i.e.

$$(-p-m)^2 + \frac{K_t}{JR_aK_b} \times (-p-m) \neq 0.$$

After several iterations, CONSOLE gets the designed parameters that meet the specifications. The position output θ plot from CONSOLE is shown in Fig. 5-1.

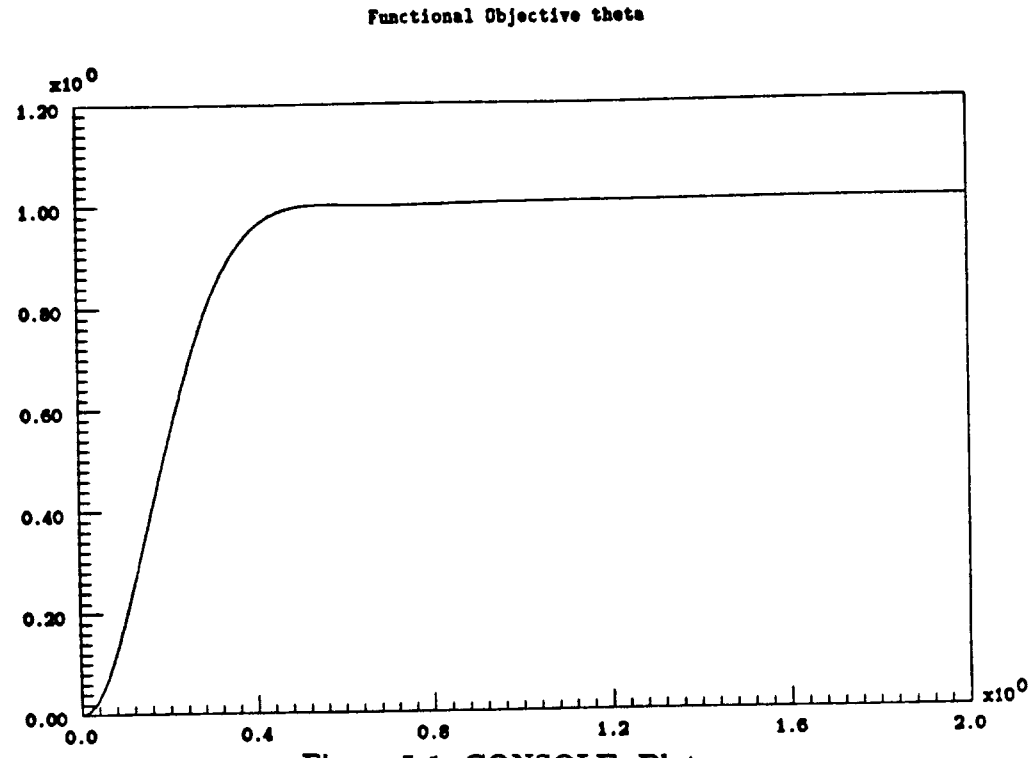


Figure 5-1 CONSOLE Plot

Listing 5-2 shows the pcomb result and the final values of design parameters.

Listing 5-2 Pcomb and Desired Design Parameters from CONSOLE

```
Pcomb(Iter=0)PRESENT  GOOD              G          B           BAD
F01 over     1.01e+00 1.01e+00 ==========|          |          1.05e+00
F02 under    9.99e-01 9.90e-01       *=========================  9.50e-01
C1  observ   1.37e+03 1.00e+00 <---------------------------- 1.00e-04
<0>
design parameter(s) for iteration 0

Name         Value
K1           8.38833e+00
K2          -1.61255e-01
K3          -1.37716e+01
m            1.33839e+01
p            2.49271e+01
```

This controller will be implemented in the real-time system with the flexible arm removed. Since there are already two sensors: position encoder and tachometer, the only state needed to be reconstructed is z. In s-domain, from Eq. 5.3,

$$Z(s) = \frac{m}{s+p} Y(s) \tag{5.8}$$

By the straight-forward sampling of Eq. 5.8, the discrete-time system is

$$z[\mathrm{kh}] = \frac{m}{p}\frac{1-e^{-ph}}{q-e^{-ph}}y[\mathrm{kh}]$$

where q is the forward shift operator and h is the sampling period. Then, with the *velocity form*[2], the state z can be found by

$$z[\mathrm{kh}] = e^{-ph}z[(\mathrm{k}-1)\mathrm{h}] + \frac{m}{p}(1-e^{-ph})y[(\mathrm{k}-1)\mathrm{h}].$$

In order to have an idea of how the controller works in the discrete form, the simulation package SIMNON is used to simulate the whole system: the continuous-time plant with a discrete-time controller. The sampling rate of the system can also be decided. Listing 5-3 is the input file for SIMNON . System CMOTOR is the plant, DMOTOR is used to reconstruct the state z and compute the feedback, MOTORCON sets up the connection between the plant and the controller.

Listing 5-3 Input File to SIMNON for Integral Control

```
continuous system CMOTOR
"Continuous-Time linear system for the motor
INPUT u
OUTPUT y1 y2
state x1 x2
DER dx1 dx2
dx1=0*x1+1*x2+0*u
dx2=0*x1-Kt/(J*Ra*Kb)*x2+K*Kt/(J*Ra)*u
y=c1*x1+c2*x2
y1=x1
y2=x2
K:6
Kt:21.62
Kb:2.443
J:0.1
Ra:33.6
c1:1
c2:0
END

discrete system DMOTOR
"Discrete-Time Integral Feedback
INPUT yr y1 y2
OUTPUT u
state i
new ni
time t
tsamp ts
u=yr-(k1*y1+k2*y2+k3*i)
ni=exp(-p*h)*i+m*(1-exp(-p*h))*y1/p
ts=t+h
k1:8.45158
```

```
k2:-1.14095e-1
k3:-1.42896e1
p:2.63678e1
m:1.37526e1
h:0.005
END

connecting system motorcon
"Connection for simulation of the motor system and integral controller
time t
yr[dmotor]=1
y1[dmotor]=y1[cmotor]
y2[dmotor]=y2[cmotor]
u[cmotor]=u[dmotor]
END
```

The simulation result for the sampling time 0.005 sec is shown in Fig. 5-2. When we increase the sampling time beyond 0.5 sec, the performance will be deteriorate. Thus the sampling rate cannot fall below 2 Hz.

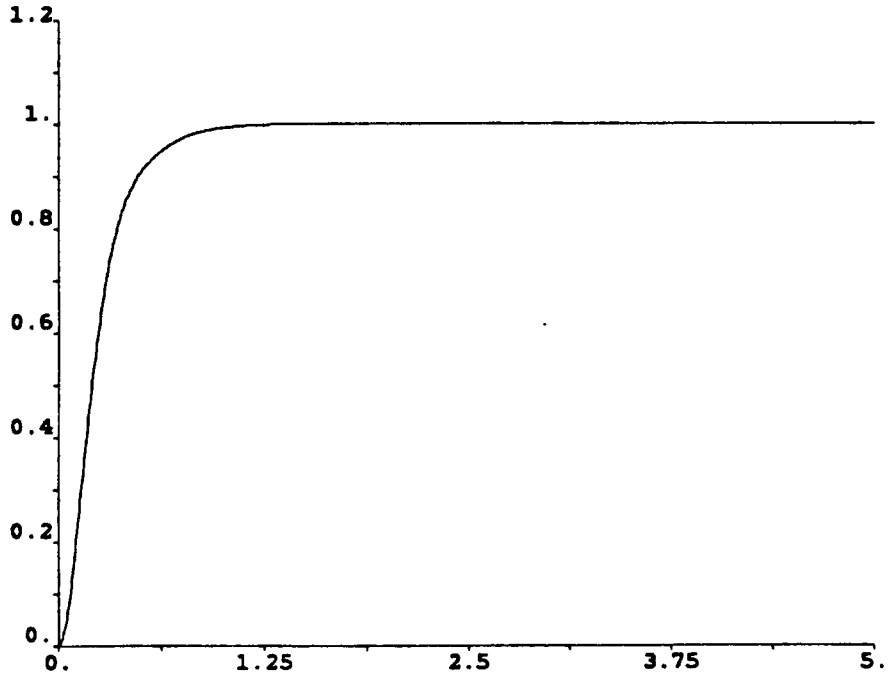


Figure 5-2 SIMNON Simulation Plot for h=0.005

With the motor system described in Chapter 2, the controller is implemented on the IBM PC/AT. The implementation result is shown in Fig. 5-3.

Due to the existence of ripple torque, this plot does not quite match the simulation result obtained from SIMNON . This problem will be discussed and

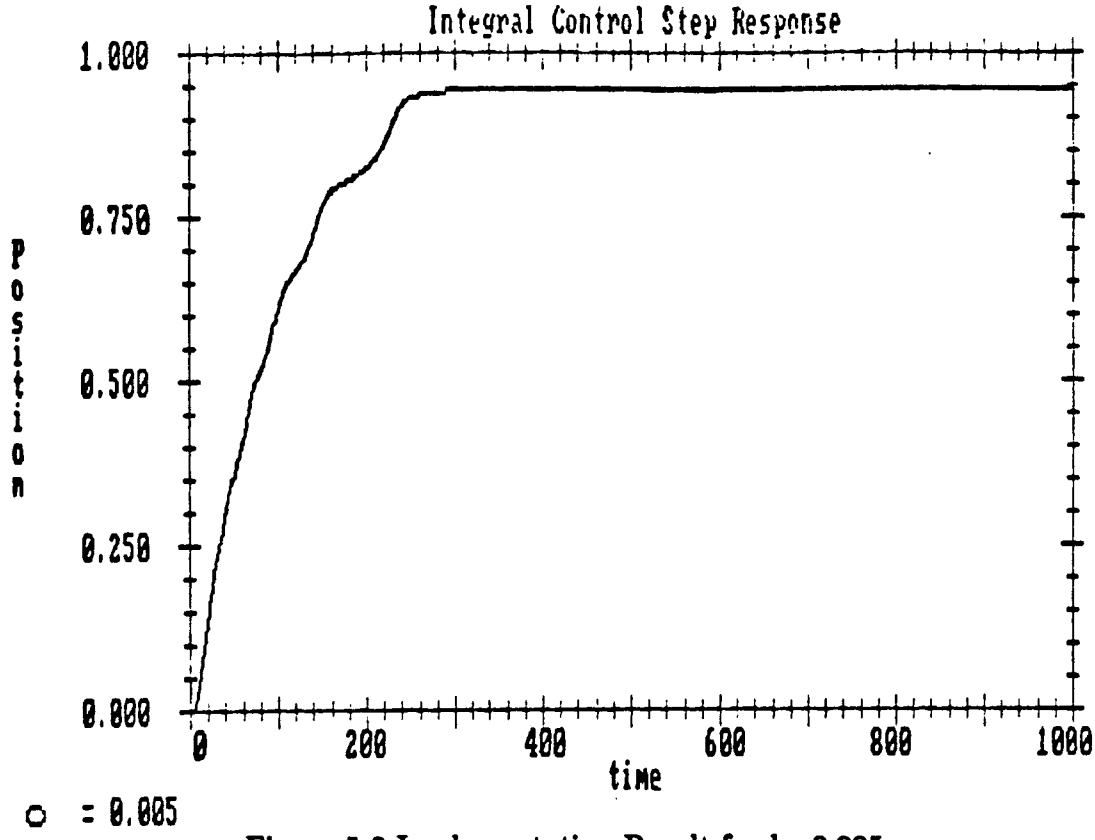


Figure 5-3 Implementation Result for h=0.005

solved later (Chapter 6) .

This simple example illustrates that the integral controller can be used to control a system. The main reason of using an integral feedback is to remove the steady state error. By the way just illustrated, it has another feature of adding more freedom (design parameters) in doing the design. This idea will be used later in designing a controller for the flexible arm.

CHAPTER VI

Identification of Actuator Characteristics

6.1 Introduction

The characteristics of an actuator (motor) in a robotics system affect the performance of a controller design in a significant way. Among the many aspects of a realistic system, there are two items of significance: friction and ripple torque. This chapter concerned with these two effects and show how to compensate for their influence.

In the extensive investigations of friction, Walrath[1] modeled the gimbal bearing friction as, in Dahl's model,

$$T_f + \tau \frac{dT_f}{dt} = (\text{sgn } \omega) T_c, \tag{6.1}$$

where T_f is the total friction, T_c is the stiction friction(a constant), and τ is a parameter to be estimated. On the other hand, Canudas[3] took the viscous friction into consideration. Here, a compromised model is set up and the experiment shows that this new model is more realistic.

6.2 Friction and Ripple Torque Modeling

6.2.1 Friction

There are essentially three kinds of frictions existing in a motor system: stiction friction, Coulomb friction, and viscous friction. The stiction friction and Coulomb friction can be taken together as bearing friction and may be modeled as in Fig. 6-1.

On the other hand, the viscous friction is coming from the motion of the drive. It is proportional to the angular velocity and can be modelled as in Fig. 6-2.

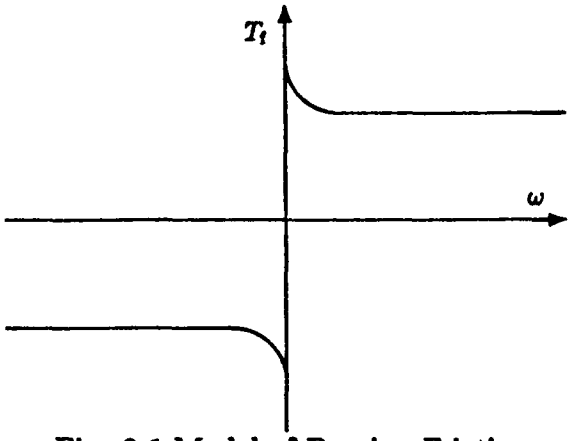


Fig. 6-1 Model of Bearing Friction

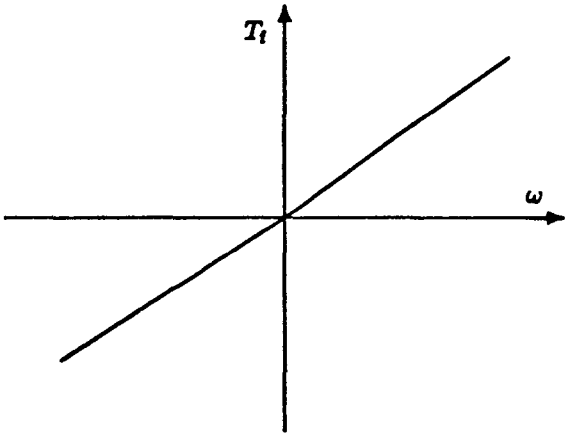


Fig. 6-2 Model of Viscous Friction

From these models, we may expect that the bearing friction will be dominant at low speed, and viscous friction will be dominant at high speed. Thus, by combining these effects, we may model the total friction as a relation of the curve as in Fig. 6-3.

The total friction torque T_f can be then represented as

$$T_f = \begin{cases} \alpha_1\omega + \beta_1, & \text{if } \omega > 0; \\ \alpha_2\omega - \beta_2, & \text{if } \omega < 0, \end{cases} \tag{6.2}$$

where α_1 and α_2 are functions of ω, β_1 and β_2 are stiction frictions for $\omega > 0$ and $\omega < 0$, respectively. In Section 6.3, the experiment is carried out to identify the parameters α and β and it is shown that the model (6.2) we described is a realistic one.

6.2.2 Ripple Torque

Due to the limitation of the number of magnetic poles in the motor, the

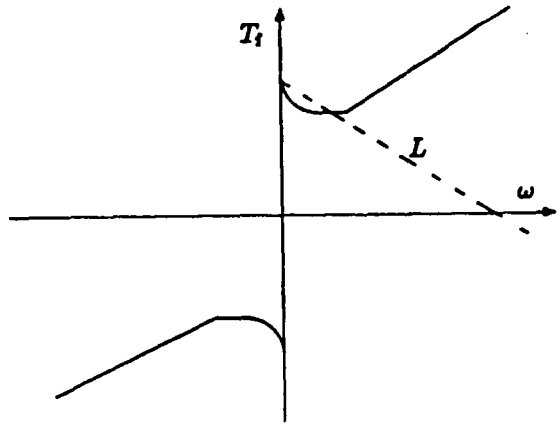


Fig. 6-3 Model of Total Friction

magnetic field is not uniform. The torque constant thus depends on the angular position of the rotor. So a periodic torque known as ripple torque exists in all permanent magnet DC motors. It will affect the system performance especially in the case of direct-drive systems. By moving the base of the motor for one revolution, one can feel the torque fluctuations. Now this problem will be investigated.

Due to the existence of the back emf constant, the motor itself is a stable system. The steady state velocity should be constant when we input a constant current. But the experiment shows that it is never a constant, instead, it is a periodic function of position. Fig. 6-4 is a plot of velocity vs. position at the steady state when we input a constant voltage.

When the velocity is low, the experiment shows this curve is almost a sinusoidal function. Therefore, the ripple torque can be modeled as

$$T_p = \kappa \sin[a(\theta + \theta_{offset})]. \tag{6.3}$$

where κ, a, θ_{offset} are to be estimated.

6.3 Experiment

6.3.1 Motor Model with Friction and Ripple Torque

With the friction and ripple torque model discussed in the previous sections, the block diagram of the whole system is shown in Fig. 6-5, which is revised from Fig. 2-2.

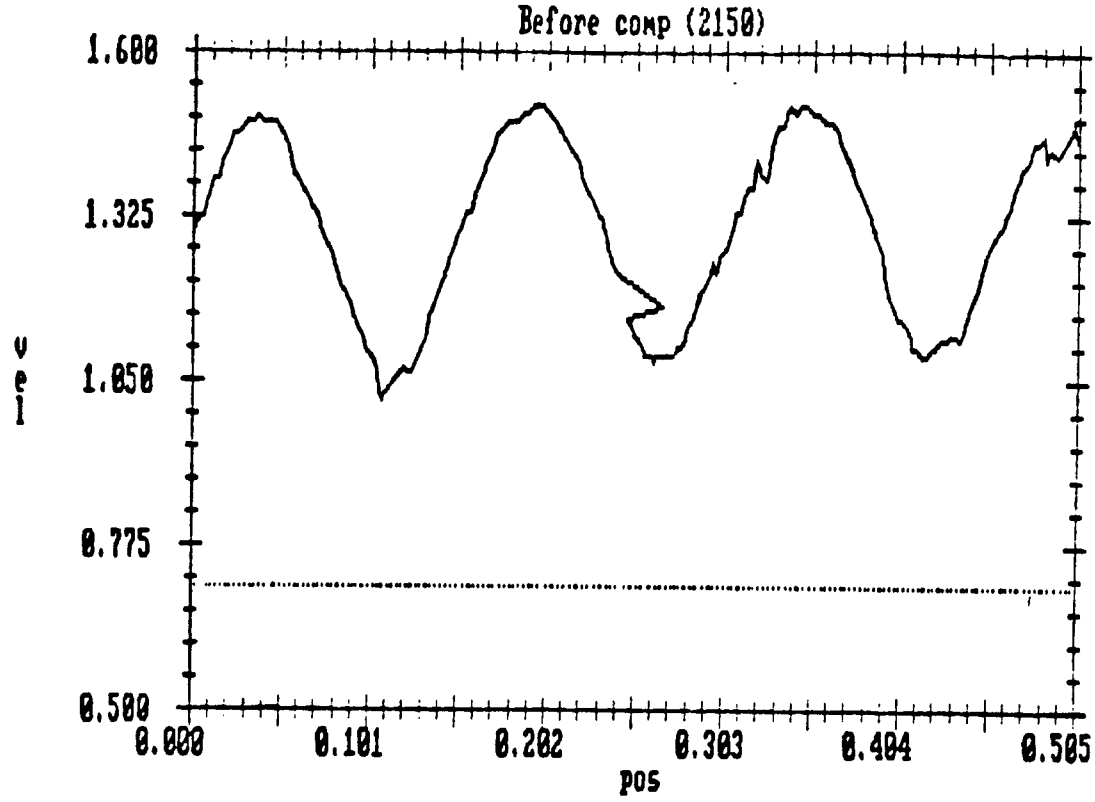


Figure 6-4 Relation between Velocity and Position

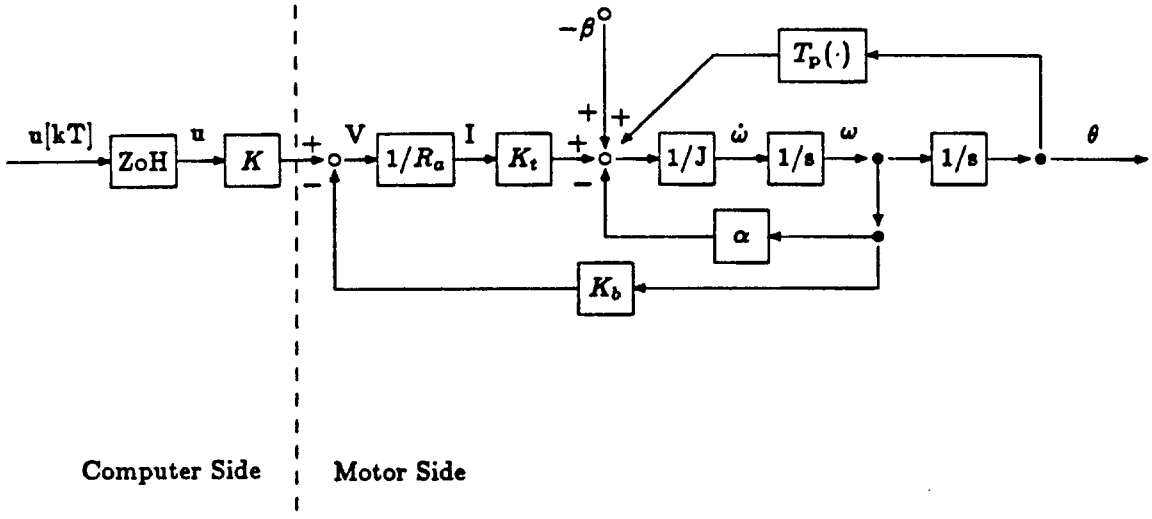


Figure 6-5 Block Diagram of the Whole System

where

α, β, κ, a, θ_{offset} as defined before, and T_p is the ripple torque.

From Eq. 2.1, the state equation of the whole system can be represented as

$$\begin{aligned} J\dot{\omega}(t) &= T(t) \\ &= \frac{K_t}{R_a}V(t) - \alpha\omega(t) - \beta + T_p(\theta) \\ &= -[\frac{K_tK_b}{R_a} + \alpha]\omega(t) + \frac{KK_t}{R_a}u(t) - \beta + \kappa\sin[a(\theta + \theta_{offset})] \qquad (6.1) \\ \dot{\theta}(t) &= \omega(t) \qquad (6.2) \end{aligned}$$

6.3.2 Friction

If we assume that there is no ripple torque, the equation is

$$J\dot{\omega}(t) = -[\frac{K_tK_b}{R_a} + \alpha]\omega(t) + \frac{KK_t}{R_a}u(t) - \beta$$

Then, at steady state, with the constant input $u(t) = \bar{u}$,

$$\omega(t) \to \bar{\omega} \quad \text{and} \quad \dot{\omega}(t) \to 0.$$

Thus,

$$0 = -[\frac{K_tK_b}{R_a} + \alpha]\bar{\omega} + \frac{KK_t}{R_a}\bar{u} - \beta.$$

Therefore, by measuring the average velocity at steady state (this take out the effect of the ripple torque), α can be computed as

$$\alpha = \frac{1}{\bar{\omega}}(\frac{KK_t}{R_a}\bar{u} - \beta) - \frac{K_tK_b}{R_a}. \qquad (6.3)$$

This estimated α is in fact the slope of the shaded line L shown in Fig. 6-5. It is not the slope of the tangent line of the T_f–ω curve.

Since the parameter β is exactly the stiction friction of the motor, it can be measured directly by applying a small voltage across the motor while the motor does not move. Record the maximum value of voltage which moves the motor; then β can be obtained.

6.3.3 Ripple Torque

If there is no ripple torque, with constant input, the steady state value of the velocity should be a constant. But the experiment shows that the steady state

velocity is a sinusoidal function of position. By measuring the peak value, frequency, and shift value, the velocity can be written as

$$\omega = \bar{\omega}\left(1 + \kappa' \sin\left[a(\theta + \theta_{\text{offset}})\right]\right) \tag{6.4}$$

Take the Laplace Transform of Eq. 6.1, without the ripple torque term, one can get

$$sJW(s) = -\left[\frac{K_t K_b}{R_a} + \alpha\right]W(s) + \frac{KK_t}{R_a}\frac{U}{s} - \frac{\beta}{s}$$

By some algebraic manipulation, we reduce this to

$$W(s) = \frac{KK_tU - R_a\beta}{K_tK_b + \alpha R_a}\left[\frac{1}{s} - \frac{1}{s + \frac{K_tK_b + \alpha R_a}{JR_a}}\right]$$

Taking the inverse Laplace Transform, one can get the step response of the velocity $\omega(t)$ as

$$\omega(t) = \frac{KK_tU - R_a\beta}{K_tK_b + \alpha R_a}\left[h(t) - e^{-\frac{K_tK_b + \alpha R_a}{JR_a}t}h(t)\right]$$

where $h(t)$ is a unit step function. Thus, as $t \to \infty$,

$$\omega(t) \to \frac{KK_tU - R_a\beta}{K_tK_b + \alpha R_a}$$

and

$$\bar{\omega} = \frac{KK_tU - R_a\beta}{K_tK_b + \alpha R_a}$$

By the formula Eq. 6.4, taking ripple torque into account, we have

$$\begin{aligned}
\omega(\theta) &= \bar{\omega}\left(1 + \kappa' \sin\left[a(\theta + \theta_{\text{offset}})\right]\right) \\
&= \frac{KK_tU - R_a\beta}{K_tK_b + \alpha R_a}\left(1 + \kappa' \sin\left[a(\theta + \theta_{\text{offset}})\right]\right) \\
&= \frac{1}{\frac{K_tK_b}{R_a} + \alpha}\left(\frac{KK_t}{R_a}U - \beta\right)\left(1 + \kappa' \sin\left[a(\theta + \theta_{\text{offset}})\right]\right) \\
&= \frac{1}{\frac{K_tK_b}{R_a} + \alpha}\left\{\frac{KK_t}{R_a}U - \beta + \underbrace{\left(\frac{KK_t}{R_a}U - \beta\right)\kappa' \sin\left[a(\theta + \theta_{\text{offset}})\right]}_{\text{Ripple Torque Term}}\right\}
\end{aligned}$$

Therefore, the ripple torque

$$T_p = \kappa \sin\left[a(\theta + \theta_{\text{offset}})\right]$$

can be found by

$$\kappa = \kappa'\left(\frac{KK_t}{R_a}U - \beta\right) \tag{6.5}$$

κ can be thus obtained by κ' and $\left(\frac{KK_t}{R_a}U - \beta\right)$.

6.4 Data Analysis

6.4.1 Friction

First, the parameter β was measured. It is essentially the stiction friction of motor. Because of the existence of the ripple torque, it is also a function of position. By averaging the voltages for starting the motor at different positions, β can be obtained as

$$\beta_1 = 1.7334 \quad \text{(Volt)} \qquad \text{for } \omega > 0$$

$$\beta_2 = 1.14 \quad \text{(Volt)} \qquad \text{for } \omega < 0$$

In order to take out the effect of ripple torque, the average value of velocity is also computed. Using the shaft encoder and the time counter on the DASH16 board, the average velocity can be obtained. Table 6.1 lists the experimental data and the calculated values of α.

Here α is computed by

$$\alpha = \frac{1}{\bar{\omega}}\frac{K_t}{R_a}(V - \beta) - \frac{K_t K_b}{R_a}$$

For the reason of different scale, this is different from Eq. 6.3. The output voltage $\bar{u}$ is changed to the input voltage to motor V. In fact, there is a gain between them, but due to the reason that there is some offset voltage in the amplifier from $\bar{u}$ to V, the latter is more suitable here.

The relationship between the velocity and α (in the unit of $lb-in/rad/sec$) is plotted in Figure 6-6.

For the reason talked before, α measured is not the tangent slope of the $T_f-\omega$ curve. The actual $T_f-\omega$ curve can be reconstructed by plotting $(\omega, \alpha\omega+\beta)$ in the $x-y$ plane. It is shown in the Fig. 6-7.

Input Voltage (Volt)	Displacement (rad.)	Time Interval (sec)	Velocity (rad/sec)	α
2.75	0.4249	0.5869	0.724	-0.6508
4.23	0.78386	0.5864	1.3367	-0.3703
5.71	1.135	0.58736	1.932	-0.2477
7.19	1.4757	0.586	2.51824	-0.177
8.61	1.8208	0.5882	3.0956	-0.1296
11.63	1.9896	0.4685	4.2467	-0.0721
14.59	2.5372	0.4689	5.4109	-0.043
17.55	2.324	0.353	6.5835	-0.026
20.5	2.7197	0.3529	7.7068	-0.005
23.5	3.11398	0.3512	8.904	0.0011
26.4	2.9406	0.2928	10	6.0153
29.4	2.637	0.2348	11.16	0.02324
32.4	2.901	0.2354	12.36	0.0245
-3.06	-0.514	0.589	-0.873	-0.1569
-4.54	-0.869	0.589	-1.478	-0.0912
-6.04	-1.2087	0.589	-2.052	-0.0354
-7.51	-1.547	0.5881	-2.6305	-0.0137
-9	-1.883	0.589	-3.198	0.0096
-11.95	-2.052	0.4713	-4.5354	0.0256
-13.42	-2.32	0.4712	-4.923	0.033
-14.90	-1.948	0.3535	-5.51	0.035
-17.85	-2.353	0.3535	-6.656	0.043
-20.8	-2.765	0.3535	-7.822	0.045
-23.8	-3.18	0.3535	-8.996	0.0489
-26.7	-3.581	0.3534	-10.148	0.0489
-29.7	-2.663	0.2358	-11.293	0.0554
-32.6	-2.938	0.2357	-12.465	0.0521

Table 6-1 Experiment Data for the Calculation of α

From the above plot, it is clear that the expected model (Figure 6.3) exactly matches the experimental result. From the values of α and β, the exact model of friction can be found and used in the compensation scheme.

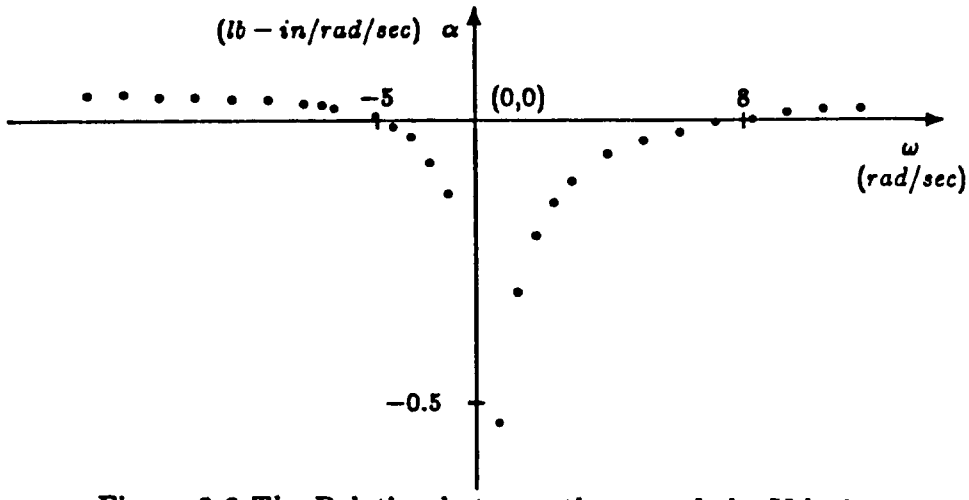


Figure 6-6 The Relation between the α and the Velocity

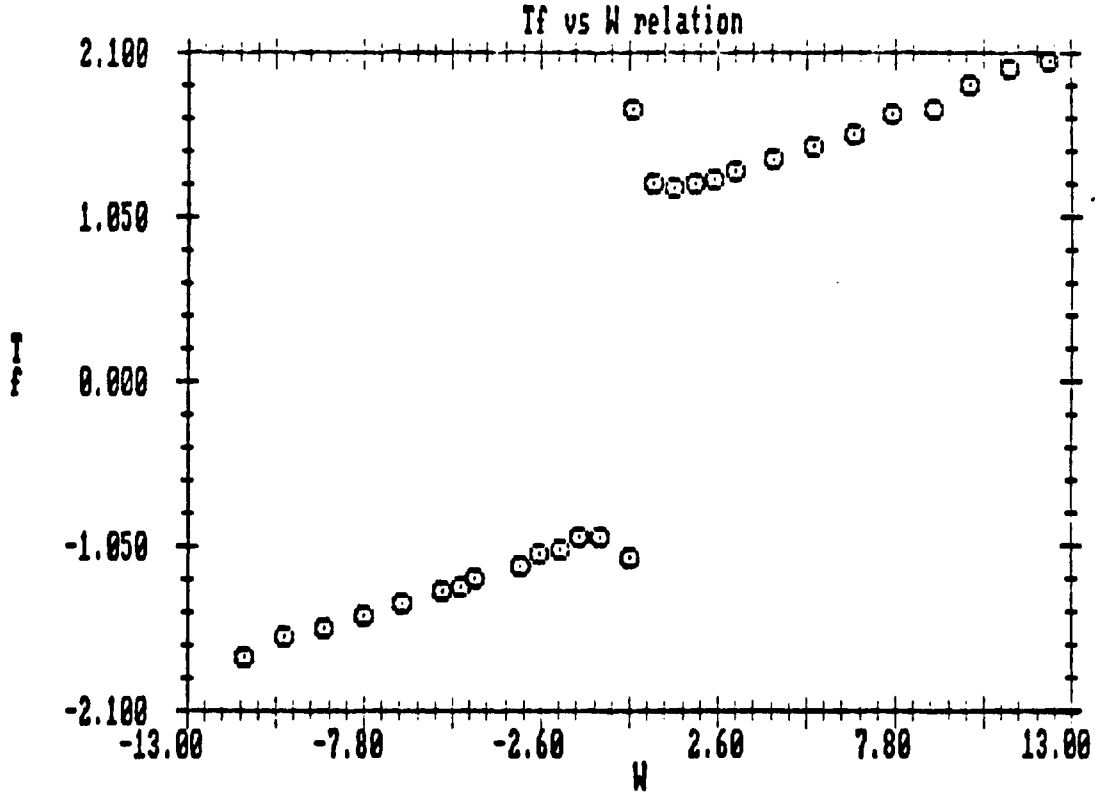


Figure 6-7 Reconstructed $T_f - \omega$ Curve

6.4.2 Ripple Torque

In Section 6.4.1, the experiment for β actually gives us an idea about the ripple torque. In order to get the whole structure of it, experiments with other inputs are

also done.

By using the tachometer, the peak value of the velocity can be found. Figure 6-4 is an example of the profile of velocity with respect to the position. As illustrated in Section 6.3.2, first the κ' in Eq. 6.5 is calculated by the formula

$$\kappa' = \frac{\text{high peak value} - \text{low peak value}}{2 \cdot \text{nominal velocity}}$$

Table 6-2 presents the experiment data for the ripple torque.

Input Voltage (Volt)	Velocity (rad/sec)			κ'
	High	Low	Nominal	
0.1807	0.885	0.16	0.523	0.693
0.205	0.905	0.25	0.578	0.567
0.254	1.025	0.385	0.7	0.457
0.498	1.485	1.07	1.277	0.163
0.742	2.02	1.665	1.8425	0.096
1.23	3.1	2.82	2.96	0.047
1.719	4.2	3.95	4.07	0.031
2.207	5.355	5.09	5.22	0.025
2.695	6.46	6.16	6.31	0.024
3.183	7.635	7.375	7.505	0.017
3.672	8.7505	8.435	8.595	0.019
4.16	9.9	9.63	9.765	0.014
4.648	11.055	10.75	10.9	0.014
5.137	12.195	11.885	12.04	0.013

Table 6-2 Experimental Data for the Ripple Torque

Figure 6-8 shows the relationship between κ' and $(\frac{KK_t}{R_a}U - \beta)$ (as in Eq. 6.5) .

Assume

$$\kappa' = c_1(\frac{KK_t}{R_a}U - \beta)^{c_2},$$

which implies

$$\log \kappa' = \log c_1 + c_2 \log(\frac{KK_t}{R_a}U - \beta).$$

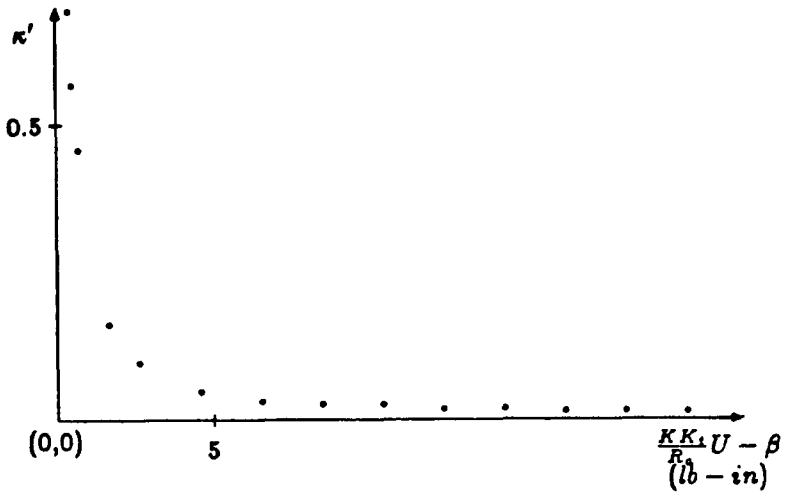


Fig. 6-8 Relation between κ' and $\frac{KK_t}{R_a}U-\beta$

By the linear regression method with the data $\log\kappa'$ and $\log\left(\frac{KK_t}{R_a}U-\beta\right)$, c_1, c_2 can be found as

$$c_1 = 0.0516$$

$$c_2 = -1$$

i.e.

$$\kappa' = 0.0516\left(\frac{KK_t}{R_a}U-\beta\right)^{-1}$$

so,

$$\left(\frac{KK_t}{R_a}U-\beta\right)\kappa' = 0.0516$$

This matches the expected formula Eq. 6.5. Thus the parameter κ of the ripple torque is

$$\kappa = 0.0516$$

Since there is some time delay between the data retrieved from the position encoder and the tachometer, the peak point shown in the velocity profile has been shifted. Thus the β-profile is used to specify the offset value. It is found that

$$\theta_{\text{offset}} = 1.509 \quad \text{(rad.)}$$

By plotting the velocity profile for a revolution, the period of ripple torque is found to be 41. Therefore, the model for the ripple torque is, finally,

$$T_{ripple} = 0.0516 \sin[41(\theta + 1.509)].$$

6.5 Compensation Scheme

With the analysis and discussion in the previous sections, now the compensator for their effects will be established.

6.5.1 Friction Compensation

The friction in a motor system may be modelled as in Fig. 6-3 as described before. Due to the fact that nonlinear functions take a lot of time in computing, an approximate piecewise linear model for the friction is chosen as

$$\hat{T}_f = \begin{cases} \gamma_1\omega + \beta_1', & \omega > \omega_1; \\ \alpha_1\omega + \beta_1, & \omega_1 \geq \omega > 0; \\ 0, & \omega = 0; \\ \alpha_2\omega - \beta_2, & 0 > \omega \geq \omega_2; \\ \gamma_2\omega - \beta_2', & \omega_2 > \omega; \end{cases}$$

where

$$\alpha_1, \alpha_2, \omega_2 < 0,$$

$$\beta_1, \beta_2, \beta_1', \beta_2', \gamma_1, \gamma_2, \omega_1 > 0,$$

are all constant. The profile of this friction model with respect to velocity is shown in Fig. 6-9. The friction torque first decreases and then increases. It means that, at first, the Coulomb friction dominates and after the critical point, the viscous friction dominates.

By using the feedback

$$I = u + \frac{\hat{T}_f}{K_t}$$

in the original model

$$J\dot{\omega} = K_t I - T_f$$

where the T_f is the true friction, the overall state equation will be

$$J\dot{\omega} = K_t u + (\hat{T}_f - T_f).$$

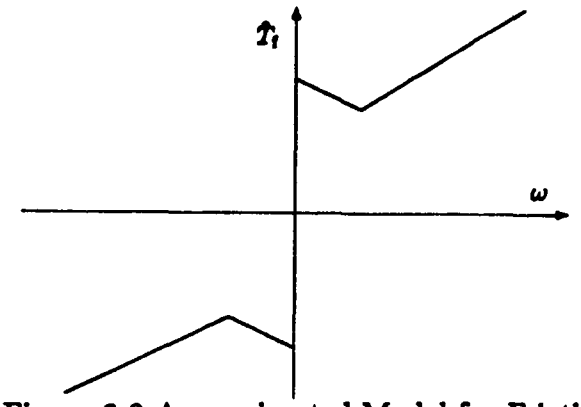


Figure 6-9 Approximated Model for Friction

Thus, if the model is "good" enough, i.e. $\hat{T}_f \approx T_f$, then

$$J\dot{\omega} = K_t u, \qquad \text{where } u \text{ is the current input.}$$

In order to show that the above friction compensator works well, a velocity tracking problem is attacked. In the following, an algorithm for speed control is first illustrated, then the compensator will be included in that algorithm. The experiment shows that this compensator makes the system response faster.

Considering the motor system described in Sec. 2.2, the state equation is

$$J\dot{\omega}(t) = \frac{KK_t}{R_a}U(t) - \frac{K_t}{R_aK_b}\omega(t). \tag{6.15}$$

Let ω_r be the reference input (velocity profile), the *set-point-on-I-only* controller [2] is used as

$$U(t) = K_p\left\{-\omega + \frac{1}{T_i}\int_0^t [\omega_r(\tau) - \omega(\tau)]\, d\tau\right\}. \tag{6.16}$$

Plug the above expression for $U(t)$ into the Eq. 6.15, to get

$$J\dot{\omega}(t) = -\left(\frac{KK_tK_p}{R_a} + \frac{K_t}{R_aK_b}\right)\omega(t) + \frac{KK_tK_p}{R_aT_i}\int_0^t [\omega_r(\tau) - \omega(\tau)]\, d\tau.$$

Taking the derivative of both sides,

$$J\ddot{\omega}(t) = -\left(\frac{KK_tK_p}{R_a} + \frac{K_t}{R_aK_b}\right)\dot{\omega} + \frac{KK_tK_p}{R_aT_i}[\omega_r(t) - \omega(t)].$$

Furthermore, taking the Laplace Transform, one can get

$$\left[Js^2 + \frac{KK_tK_bK_p + K_t}{R_aK_b}s + \frac{KK_tK_p}{R_aT_i}\right]W(s) = \frac{KK_tK_p}{R_aT_i}W_r(s).$$

Thus, the transfer function from ω_r to ω can be written as

$$G(s) = \frac{\frac{KK_tK_p}{R_aT_i}}{Js^2 + \frac{KK_tK_bK_p + K_t}{R_aK_b}s + \frac{KK_tK_p}{R_aT_i}}$$

With this continuous linear system and letting K_p, T_i be design parameters, CONSOLE may be used to do the design (tuning the gains.)

Appendix B lists the input file for CONSOLE in this problem. Instead of using the time-domain representation, the frequency-domain representation (transfer function) is used in the system description file.

The values of design parameters and the *Pcomb* output are shown below.

```
Name        Value         Variation  wrt 0   Prev    Iter=0
Kp          1.07312e+00   1.0e+00
Ti          2.03349e-01   1.0e+00

Pcomb (Iter=  0) (Phase 2) (eps= 1.000e+00) (MAX_COST_SOFT= 0.807298)

SPECIFICATION     PRESENT   GOOD                 G          B              BAD
F01  speed        1.00e+00  1.01e+00 ********    |          |           1.05e+00
F02  undershoot   9.17e-01  9.90e-01             |       ************* 9.00e-01
```

The step response of the closed-loop system plotted by CONSOLE is shown in Fig. 6-10.

After getting these parameters, the next thing is to translate the controller into discrete-time. The s–domain equation, from Eq. 6.16,

$$U(s) = -K_pW(s) + \frac{K_p}{T_is}[W_r(s) - W(s)]$$

may be transformed into

$$u[\mathrm{kh}] = -K_p\omega[\mathrm{kh}] + \frac{K_p}{T_i}\frac{h}{q-1}(\omega_r[\mathrm{kh}] - \omega[\mathrm{kh}])$$

by using Euler Approximation in the integral part. In order to avoid the windup (integrator saturation) problem [2], the *velocity form* is chosen in getting the control

$$u[\mathrm{kh}] = u[(\mathrm{k}-1)\mathrm{h}] - K_p\{\omega[\mathrm{kh}] - \omega[(\mathrm{k}-1)\mathrm{h}]\} + \frac{K_ph}{T_i}\{\omega_r[(\mathrm{k}-1)\mathrm{h}] - \omega[(\mathrm{k}-1)\mathrm{h}]\}$$

with $u[0] = 0$.

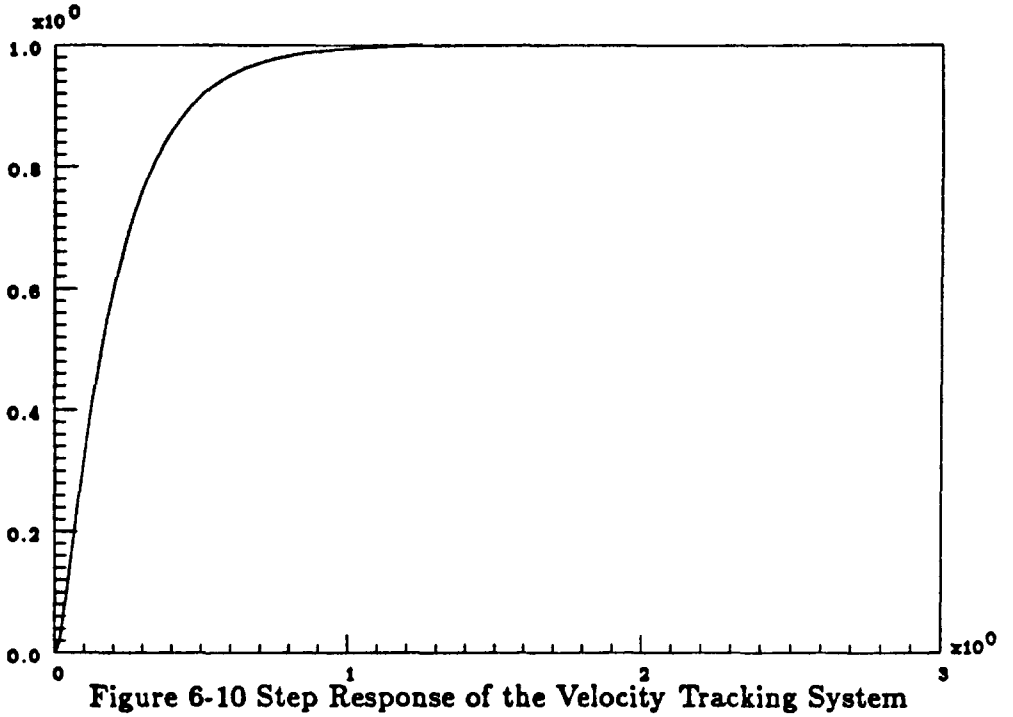


Figure 6-10 Step Response of the Velocity Tracking System

Then SIMNON is used to simulate the whole system. The input file for it is also listed in Appendix B. By choosing the sampling time to be 0.002 sec, the simulation plot is shown in Fig. 6-11.

Next step is doing the implementation. The real-time program was written in C language. Fig. 6-12 gives the experimental comparison of controllers with and without friction compensation. The sampling period is 0.002 sec (i.e. for each iteration.) The scheme with the friction compensator really improves the performance.

6.5.2 Ripple Torque Compensation

From Sec. 6.4.3, the ripple torque T_p can be modelled as

$$T_p = 0.0516 \sin[41(\theta + 1.509)].$$

Thus, by using the compensator

$$u(t) = v(t) - \frac{R_a}{KK_t}\left(0.0516 \sin[41(\theta + 1.509)]\right)$$

in Eq. 6.4, the ripple torque can be reduced.

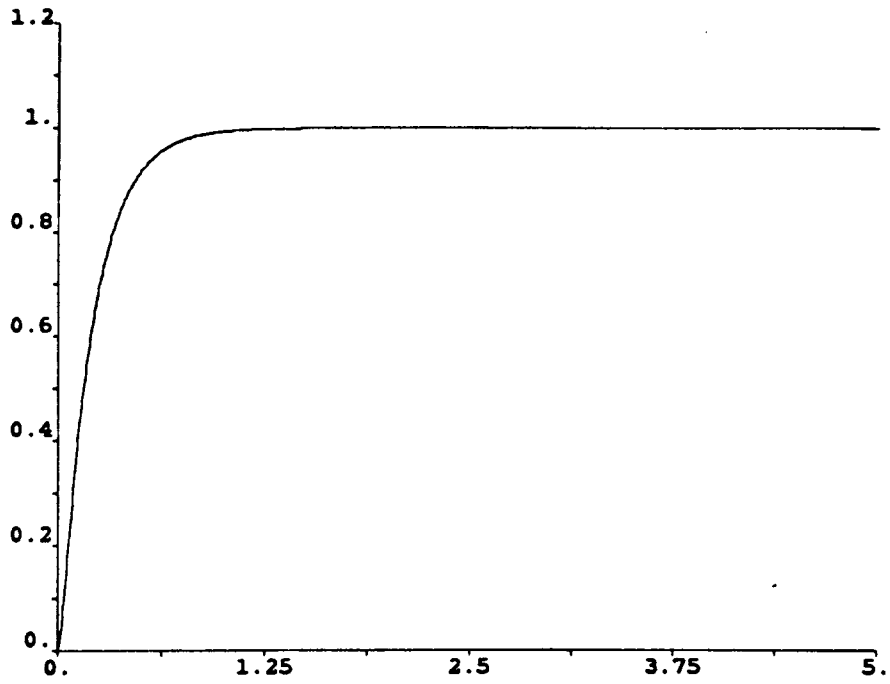


Figure 6-11 Simulation by SIMNON for the Velocity Tracking Problem

But due to the computing time between getting data from the position encoder and issuing the control signal, there is a *computational delay*[18] . In order to handle this problem, it is necessary to measure the computing time accurately. In Sec. 7.2, the computing times for some functions in Microsoft C have been compared. Based on that discussion, the code may be "optimized" in the sense of shortest computing time. The compensator is thus implemented on the IBM PC in a Microsoft C program as shown below.

```
sine = sin( 41 * (posdata[i] + time * veldata[i] - offset) );
Volt[i] = motref - k * sine;
motinp = Volt[i] * 204.8 + 2006;
dda6( motinp );
```

Here `time` is the measured time period between obtaining position data and sending the control signal. Fig. 6-13 shows the plot of velocity-vs-position after compensation. Compared with Fig. 6.4, it shows that the ripple torque problem can be solved using explicit compensation.

Since the effect of ripple torque is reduced as the load inertia increases, for a heavy arm, it can be neglected. But for a light-weight arm, as in our case, it should be properly compensated for. Sec. 7.3 will include the compensator in the control of

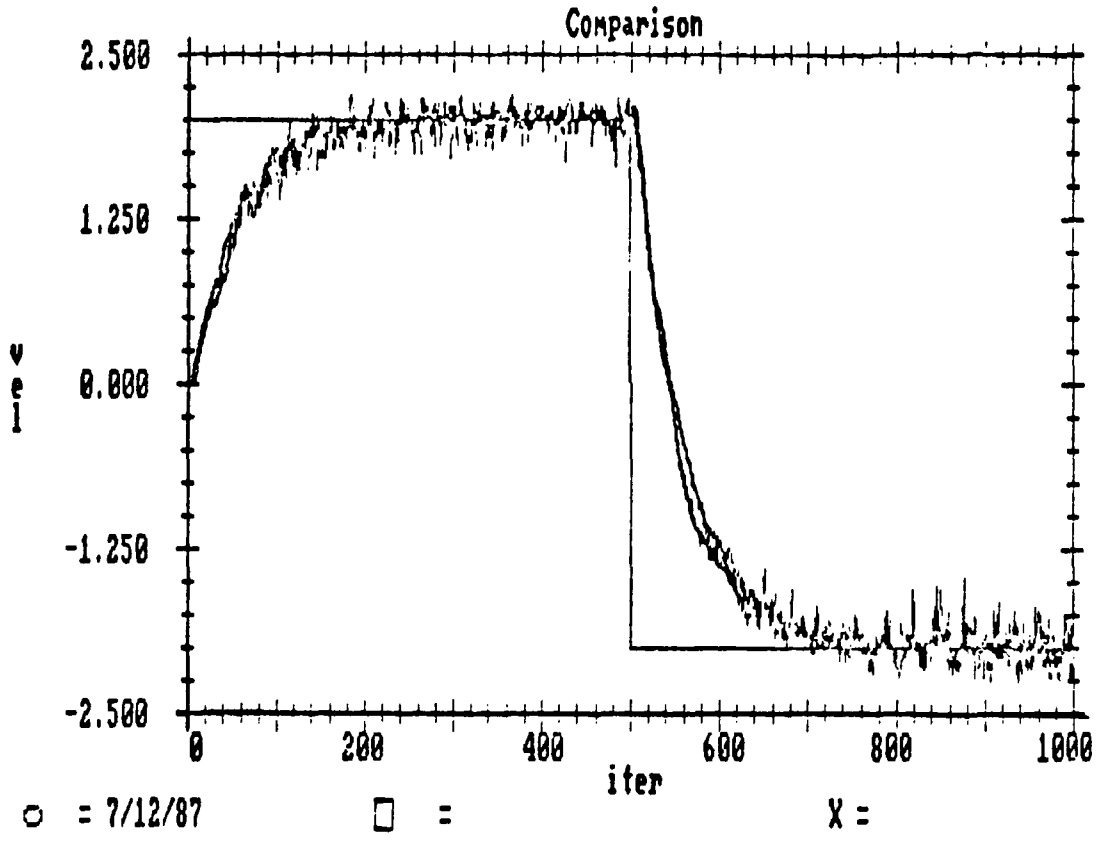


Figure 6-12 Comparison between Two Schemes in the Velocity Tracking Problem

a flexible arm. The various constants are to be re-estimated in the new environment.

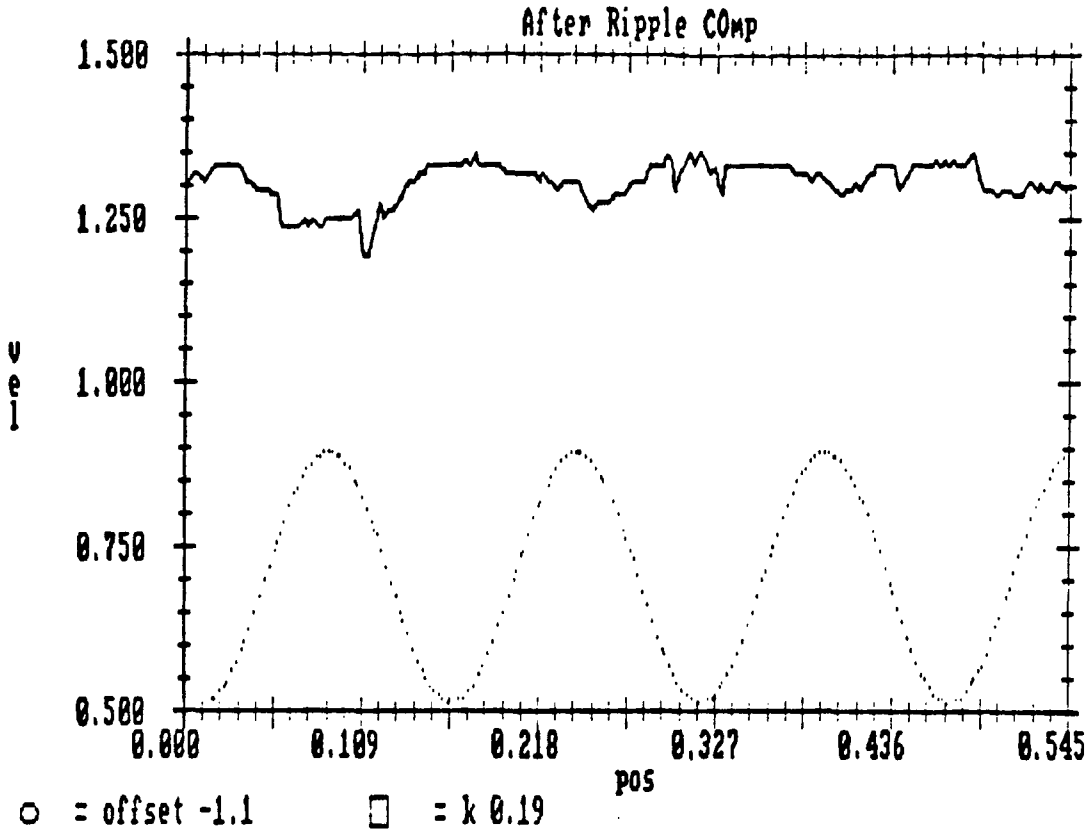


Figure 6-13 Velocity vs Position Profile after Ripple Torque Compensator

CHAPTER VII

Implementation

7.1 Hardware

7.1.1 D/A and A/D Converter

The D/A(digital-to-analog) converter chosen in the experiment is the DDA06 board manufactured by Metrabyte Company. DDA06 is an analog/digital I/O expansion board for the IBM PC. It provides six channels of 12 bit analog output and 24 lines of digital I/O. The 8255 programmable peripheral interface chip is used for digital I/O and can be operated in any mode (straight I/O, strobed I/O, and bidirectional I/O). In this experiment, one D/A channel is used to send a voltage signal (+10/-10 Volt) to motor servo electronics and 16 lines of digital I/O are used to get the 12 bit data from the position encoder. The I/O operations of DDA06 are essentially the port (address) manipulations. The function

```
dda6( data )
int data;
{
   int basedda6, chan;
   int xh, xl;
   basedda6 = 768;
   xh = data/256;
   xl = data - xh * 256;
   outp( basedda6, xl);
   outp( basedda6+1, xh);
   return;
}
```

sends a voltage through DDA06 by an integer between -2048 and 2047, where basedda6 is the base address of DDA06 board. The lines

```
   outp( basedda6+15, 0xbe );
/*****   Input position data   *****/
   pa = inp( basedda6+12 );
   pb = inp( basedda6+13 );
   posdata = ((pb*256+pa) >> 4)*3.141592653 / 2048;
```

set the operation mode 1 for the 8255 chip and read position data (from the shaft

encoder) in the unit of *rad*.

Another of Metrabyte's products, DASH16 is used as the A/D (analog-to-digital) converter. It provides eight differential channels each with a 12 bit successive approximation converter with a 25 μs conversion time, a 3 channel programmable interval timer (INTEL8253), and some other features (see [6]). In our experiment, two analog signals ranging over +10 to −10 (Volt) from the tachometer and the accelerometer are converted to integer numbers ranging over +2047/-2048 by the two channels in DASH16. The timer on DASH16 is used to set the sampling period for the control loop. With the software written by manufacturer and the interface which will be discussed later, the DASH16 board can be easily used in the programming. The following is a subroutine of initializing the DASH16 board and also the counter inside it.

```
initd16()
{
   mode = 0;
   d16io[0] = based16;
   d16io[1] = 7;
   d16io[2] = 1;
   flag = 0;
   dash16( &mode, d16io, flag );
   if( flag > 0 ) error( flag, mode );
/*****   Setup the Counter 0   *****/
   outp( based16+10, 2);    /* Set the C1=1 C0=0 */
   mode = 10;
   d16io[0] = 2;   /*   Configuration 2 */
   flag = 0;
   dash16( &mode, d16io, &flag );
   if ( flag > 0 ) error( flag, mode );
   mode = 13;
   d16io[0] = 4;    /*   Write OP2 = 1      */
   dash16( &mode, d16io , &flag );
   if ( flag > 0 ) error( flag, mode );
}
```

7.1.2 Handshaking

In order to set up the connection between the programmable peripheral interface 8255 residing on the DDA06 board and the encoder, it is necessary to set up a handshaking scheme. The strategy is as follows.

1. Program sends a RD (ReaD) signal to 8255. (e.g. by the `inp()` function call in C)
2. 8255 sets the IBF (Input Buffer Full) to be low and also send a BICTS (BInary Clear To Send) signal to the encoder.

3. After receiving BICTS from 8255, encoder sends a BISTROBE (BInary STROBE) signal to 8255 and sets the STB (STroBe) of 8255 to be high and then loads data into 8255
4. 8255 sends the data to the variable in the program and drives the IBF to high, and also STB.

The operation mode 1 (strobed I/O) of 8255 is used. The configuration between 8255 and the encoder servo electronics is shown in Fig. 7-1. The timing between those signals is shown in Fig. 7-2. With this scheme, the program can get the shaft encoder data without any error.

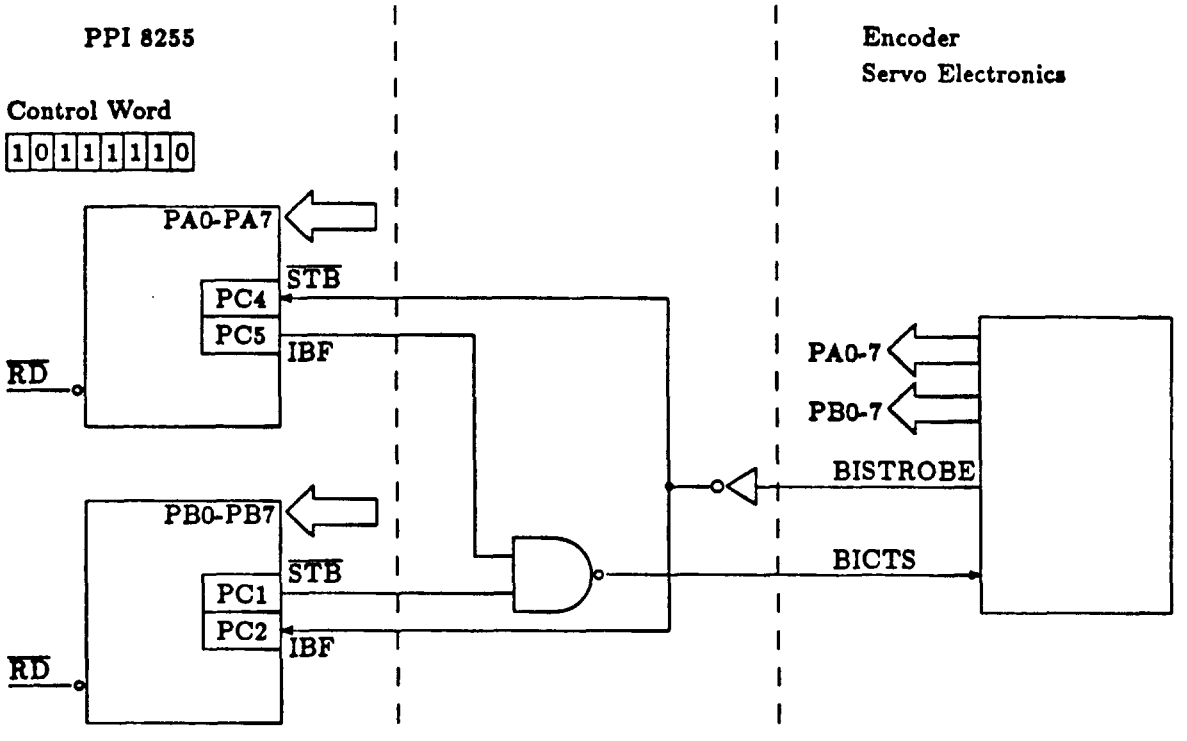


Figure 7-1 Configuration between Encoder and PPI8255

7.1.3 Low-Pass Filter

Since there is high frequency noise coming from the analog sensors, i.e. the tachometer and the accelerometer, it is necessary use low-pass filters. On the other hand, because the differential signals from the sensors do not have the same ground with the DASH16 board, it is also necessary to design a voltage follower (buffer) for

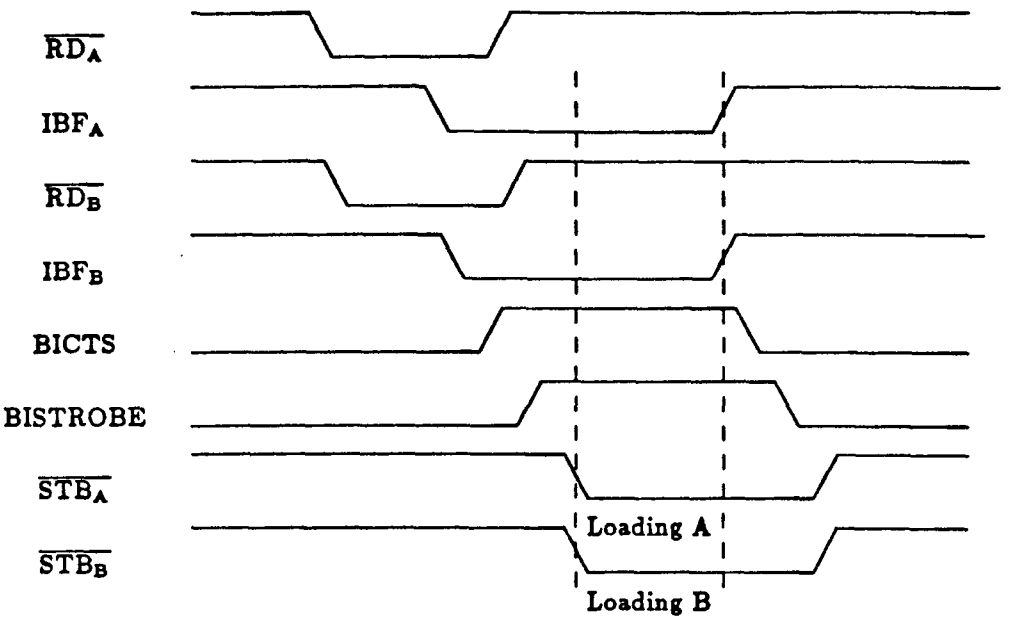


Figure 7-2 Timing

the differential signal. The circuit diagram for the filter and the buffer is shown in Fig. 7-3. The potentiometer is used to adjust the offset value of the amplifiers.

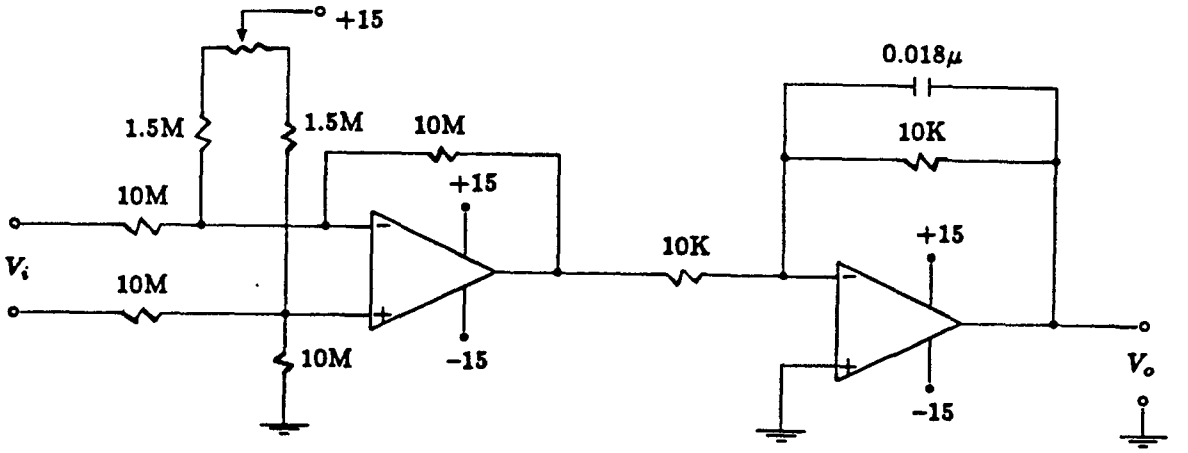


Figure 7-3 Low-pass Filter and Voltage Follower

The cutoff frequency is

$$\frac{1}{2 * \pi * 10000 * 0.018 * 10^{-6}} = 884 \text{ (Hz)}$$

From the experiment, it is found that the errors from the sensors reduced from ±11 bits to ±3 bits.

7.2 Software

7.2.1 Interface between DASH16 and C language

For the DASH16 board, there is a software package DASH16.OBJ designed by the manufacturer which is originally written in BASIC. In order to use it within a C program, two problems have to be solved:

1) The Microsoft-C compiler automatically adds an underscore "_" before any function call when it generates the object code for a C routine. For BASIC, there is no such addition.
2) The way to process the argument of a function is different. For C, the arguments are pushed into a stack from left to right, but for Basic, the order is from right to left.

Thus it is necessary to write an Assembly language subroutine to set up the connection. The C programs then call the assembly routine, and the assembly routine calls the object code DASH16.OBJ.

The following List 7-1 is the listing of the assembly interface code. It includes a named "_dash16" function which calls the function "dash16."

List 7-1 Listing of Interface Assembly Code

```
        TITLE interface
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS

DGROUP  GROUP CONST, _BSS, _DATA
        ASSUME CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP
EXTRN   DASH16:FAR
_TEXT   SEGMENT
        PUBLIC _dash16
_dash16 PROC NEAR
        push bp
        mov bp, sp
        xor ax, ax
        mov ax, [bp+4]
        push ax
        mov ax, [bp+6]
        push ax
        mov ax, [bp+8]
        push ax
```

```
        call DASH16
        mov sp, bp
        pop bp
        ret
_dash16 ENDP
_TEXT   ENDS
END
```

7.2.2 Computing Time

In a real-time system, the computing time is crucial in design. In order to have some idea of how much time is spent in evaluating some specific functions, the computing time for them has been estimated by using the timer counter on the DASH16 board. Table 7.1 gives the computing time for some commands in C language. It reveals some basic facts in real-time computing.

Integer Operations	Time (μs)	Floating Point Operations	Time (μs)
Assignment		Assignment	
Constant	2.1	Constant	47.5
Variable	small	Variable	8
Addition(+)		Addition(+)	
Constant	0.1	Constant	0.5
Variable	0.4	Variable	9
Multiplication(×)		Multiplication(×)	
Constant	0.1	Constant	0.5
Variable	0.6	Variable	10
Shift(>>)	0.4	Functions	
Modulus(%)	0.6	printf()	66500
Bitwise AND(&)	0.3	sin()	874
Bitwise Exclusive OR(^)	0.3	asin()	603
Functions		sinh()	801
sizeof()	2.2	pow()	971
abs()	19.4	fabs()	35
Port I/O		exp()	600
outp()	17	log()	539
		sqrt()	187

Table 7.1 Computing Time for some C Function

* Except for the assignment command, the time used for variable operations is much more than for constant operations. So use constant operation if possible.
* If the assignment command is desired for constant, e.g. $T = 0.012$, it is better to define that constant by another variable outside the control loop, e.g. $S = 0.012$, then use $T = S$ in the control loop.
* I/O functions cost a considerable penalty in time. Avoid them in the control loop in any case.
* Integer operations always take very little time. Use them rather than floating point operations whenever possible.

These "facts of life" should be kept in mind in the real-time programming.

7.3 Controller Implementation for a Flexible Arm

Following the insights of Chapter 6, an integral controller-observer scheme was implemented for control of the flexible arm, with the system described in Section 2.1. The design procedure, as illustrated in the Chapter 1, is as follows,

(1) designing the state-integral feedback gain by using CONSOLE
(2) translating the continuous-time design to discrete-time
(3) designing a discrete-time observer
(4) simulating the whole system using SIMNON
(5) implementing the scheme in real-time system
(6) adding the compensator for friction and ripple torque

Besides the step (3), which exactly follows the discussion in [11], each step will be described in the following sections.

7.3.1 Designing the state-integral feedback gain by using CONSOLE

As discussed in Sec. 4.2 and Chapter 5, CONSOLE is used to design the feedback gain from each state of the system and the integral of the plant output. In order to preserve the controllability and observability properties, the conditions in Proposition 5-1 and 5-2 should be satisfied. Since here an observer is used to reconstruct the states, these constraints are important. Appendix C lists the input file for CONSOLE . The constraints named "control" and "observ" are used for the purpose mentioned above.

The gains designed from CONSOLE and *Pcomb* output are listed as follows.

```
Name        Value        Variation  wrt 0   Prev   Iter=9
k1          8.23494e+00  1.4e+00
k2         -1.58345e-01  9.0e-02
k3         -1.57999e-03  8.4e-04
k4          2.38706e-05  1.7e-05
k5          4.08994e+01  4.3e+01
k6          5.85300e-01  2.0e-01
k7         -2.38586e-06  1.0e-06
m           1.21241e+01  5.0e+00
p           1.07714e+01  3.0e+01

Pcomb (Iter= 9) (Phase 2) (eps= 1.000e+00) (MAX_COST_SOFT= 0.934643)

SPECIFICATION    PRESENT   GOOD              G          B            BAD
O1  steadystat  6.63e-08  0.00e+00 ==========*          |           1.00e-04
FO1 over-shoot  1.09e+00  1.01e+00 ====================*|           1.10e+00
Fo2 settling    9.91e-01  9.90e-01          *************************  9.50e-01
C1  observ1     1.07e+00  1.00e-02 <---------------------------------  1.00e-04
C2  observ2     1.02e+00  1.00e-02 <---------------------------------  1.00e-04
C3  observ3     6.16e-01  1.00e-02 <---------------------------------  1.00e-04
C4  control1    7.04e+00  1.00e-02 <---------------------------------  1.00e-04
FC1 max-accel   2.71e+00  1.00e+01 <--       |          |            1.10e+01
FC2 max-accel  -6.71e-01 -1.00e+01 <---------------------------------- -1.10e+01
FC3 control     1.03e+00  4.00e+00 <--       |          |            5.00e+00
```

The simulation of the continuous system with those gains is shown in Fig. 7-5.

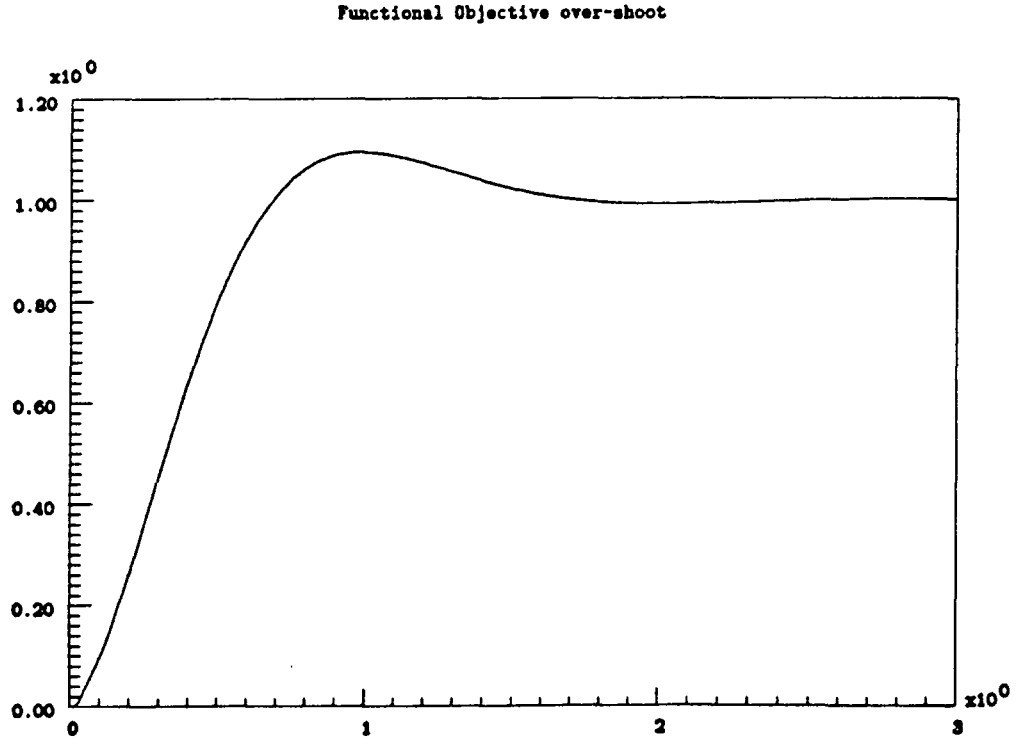


Figure 7-5 Simulation of the Flexible Arm System from CONSOLE

7.3.2 Translating the Continuous-time Design to Discrete-time

A linear time-invariant state equation may be written as

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

$$y = Cx$$

This continuous-time equation can be translated to discrete-time as

$$x[(k+1)h] = \Phi x[kh] + \Gamma u[kh]$$

$$y[kh] = Cx[kh]$$

where

$$\Phi = \exp(Ah)$$

$$\Gamma = \int_0^h \exp(A\tau)\, d\tau\, B$$

In finding the exponential of a matrix, many methods can be used. Since just one computation is needed here, the most straightforward way is chosen, i.e. the series expansion of it,

$$\Phi = I + Ah + \frac{A^2h^2}{2!} + \frac{A^3h^3}{3!} + \cdots.$$

By using a simple routine in MACSYMA as

```
noloop:100;
EXPA:ident(N);
AH:ident(N);
for i:1 step 1 thru noloop do
   (AH:h*A.AH/i,
    EXPA:EXPA+AH
   );
EXPA;
```

Φ can be computed as accurately as we need.

Similarly, Γ may be obtained from

$$\Gamma = \int_0^h (I + A\tau + \frac{A^2\tau^2}{2!} + \frac{A^3\tau^3}{3!} + \cdots)\, d\tau\, B$$

$$= Bh + \frac{ABh^2}{2!} + \frac{A^2Bh^3}{3!} + \cdots.$$

Also, a routine in MACSYMA of the form

```
noloop:100;
Gamma:h*B;
AB:h*B;
for i:1 step 1 thru noloop do
   (AB:h*A.AB/(i+1),
```

```
    Gamma:Gamma+AB
   );
Gamma;
```

may be used to compute Γ.

Other packages, such as MATLAB, CC, have also been tried and we have found that the method used here is the more reliable one.

In translating the feedback law

$$u = -Kx + K_I \int y$$

to discrete-time, the same structure used in Sec. 5.4 is chosen here for the integral part. For the state feedback part, the formula in [2] is used as

$$\tilde{K} = K[I + (A - BK)\frac{h}{2}].$$

Thus, the discrete feedback law is as

$$u[\mathrm{kh}] = -\tilde{K}x[\mathrm{kh}] + K_I z[\mathrm{kh}]$$

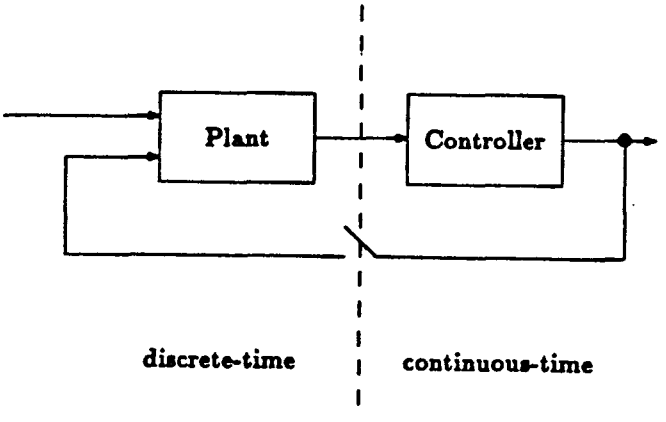
with the formula for z

$$z[\mathrm{kh}] = e^{-ph}z[(\mathrm{k}-1)\mathrm{h}] + \frac{m}{p}(1 - e^{-ph})x_1[(\mathrm{k}-1)\mathrm{h}],$$

since x_1 is exactly the tip position.

7.3.3 Simulating the Whole System by Using SIMNON

The structure



is used to do the simulation with SIMNON , where the plant is the continuous system and controller includes the discrete-time observer and the feedback law. The input system description file for SIMNON is listed in Appendix C. With the sampling rate of 100 Hz, the tip position profile is shown in Fig. 7-6.

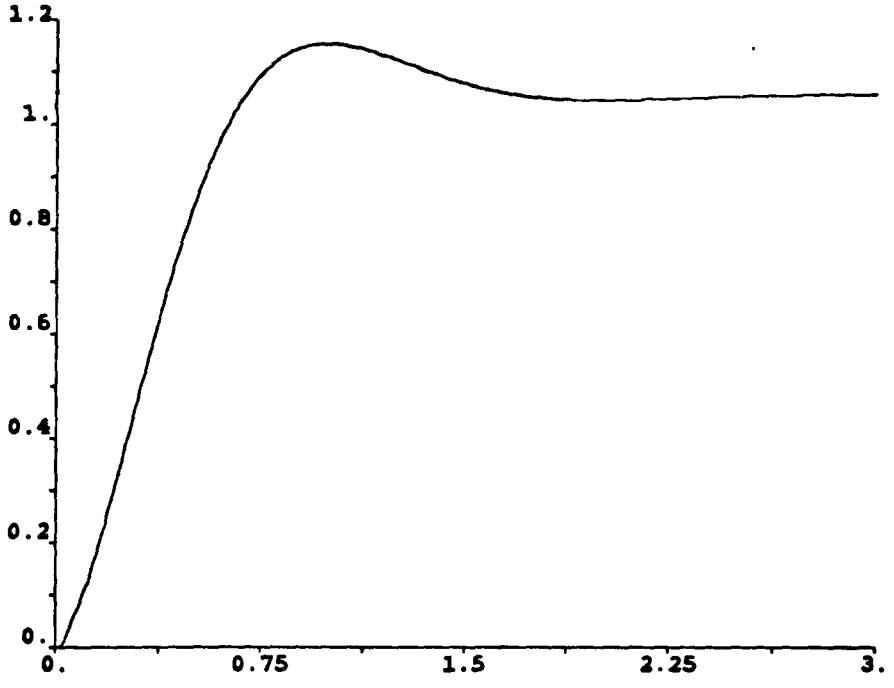


Figure 7-6 Simulation with the Whole System from SIMNON

By choosing different sampling rate with SIMNON , it shows that the sampling rate cannot be below 50 Hz. Needless to say, for different sampling rates, the discrete-time observer used is different.

7.3.4 Implementing the Scheme in Real-time System

The controller discussed before has been implemented in the real-time system with the controller residing in an IBM PC/AT. The whole system is shown in Fig. 2-1. Some key points used in the real-time programming have been discussed before. One other thing need to be noted is that, for the reason that the saturation problem in the D/A converter and amplifier, a saturator is used to the input signal to the observer [18] as

```
u_high = 2000;
u_low = -2000;
```

```
if( u_force < u_low )        u_force = u_low;
else if( u_force > u_high ) u_force = u_high;
else                         ;
```

With the software written in Microsoft C (the code is listed in Appendix D), the hub position profile from the shaft encoder is shown in Fig. 7-7.

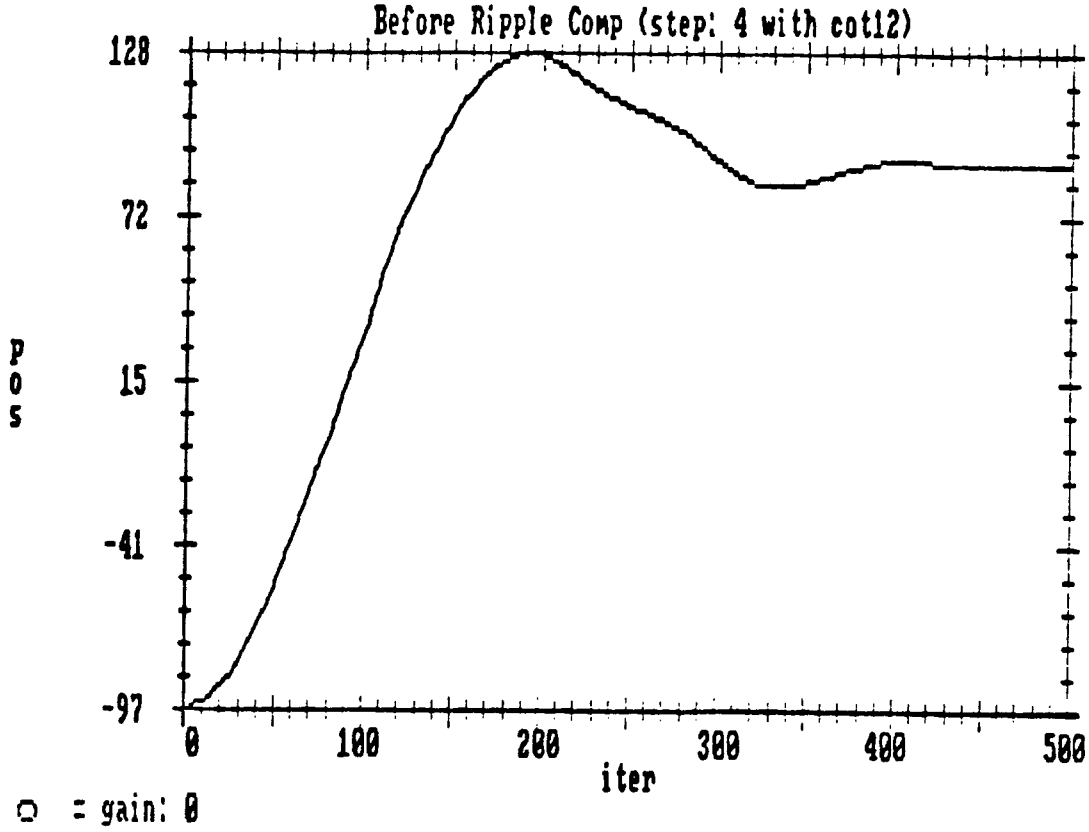


Figure 7-7 Hub Position Profile from Position Encoder

7.3.5 Adding the Compensator for Friction and Ripple Torque

The compensators for friction and ripple torque have also been added to the controller. The new model for ripple torque (since the load inertia has changed with the addition of the arm), is now

$$T_p = 0.18 \sin[41(\theta - 0.62)].$$

Fig. 7-8 shows the performance of the integral control with the ripple torque compensator. Since that compensator will add as well as reduce the speed during

the motion, the rise time was not changed much. But, when the beam settles down, the settling time and the steady-state error are both reduced much.

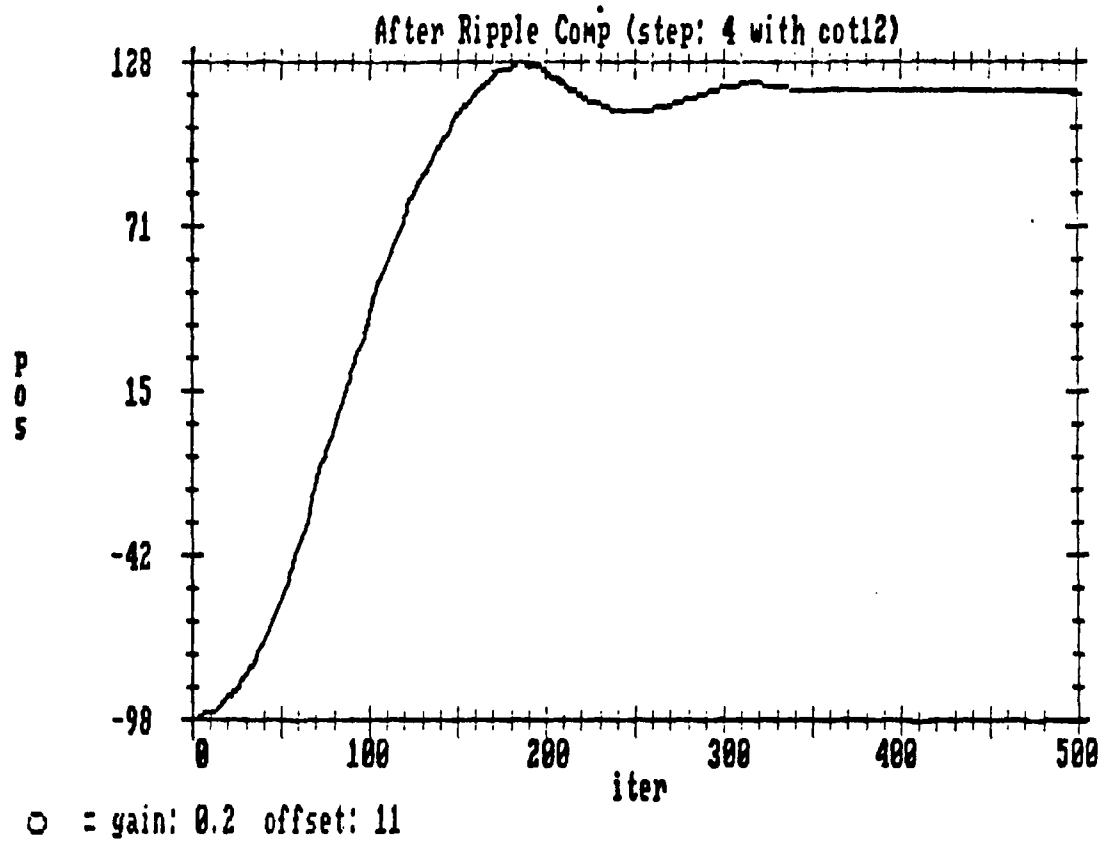


Figure 7-8 Hub Position Profile after Compensation for Ripple Torque

The friction with the flexible arm attached can be modeled as

$$\hat{T}_f = \begin{cases} 0.02\omega + 40, & \omega > 1.12; \\ -0.115\omega + 59, & 1.12 \geq \omega > 0; \\ 0, & \omega = 0; \\ -0.5\omega - 41, & 0 > \omega \geq -1.04; \\ 0.021\omega - 30, & -1.04 > \omega; \end{cases}$$

Fig. 7-9 shows the performance of the integral controller with both compensators. The improvement is quite evident. The rise time is reduced from 0.6 sec to 0.4 sec and there is almost no overshoot.

The experimental results shows that the characteristics of the actuator are really important in the design of the controller of a robot, especially as in our case, a direct-drive low-inertia system.

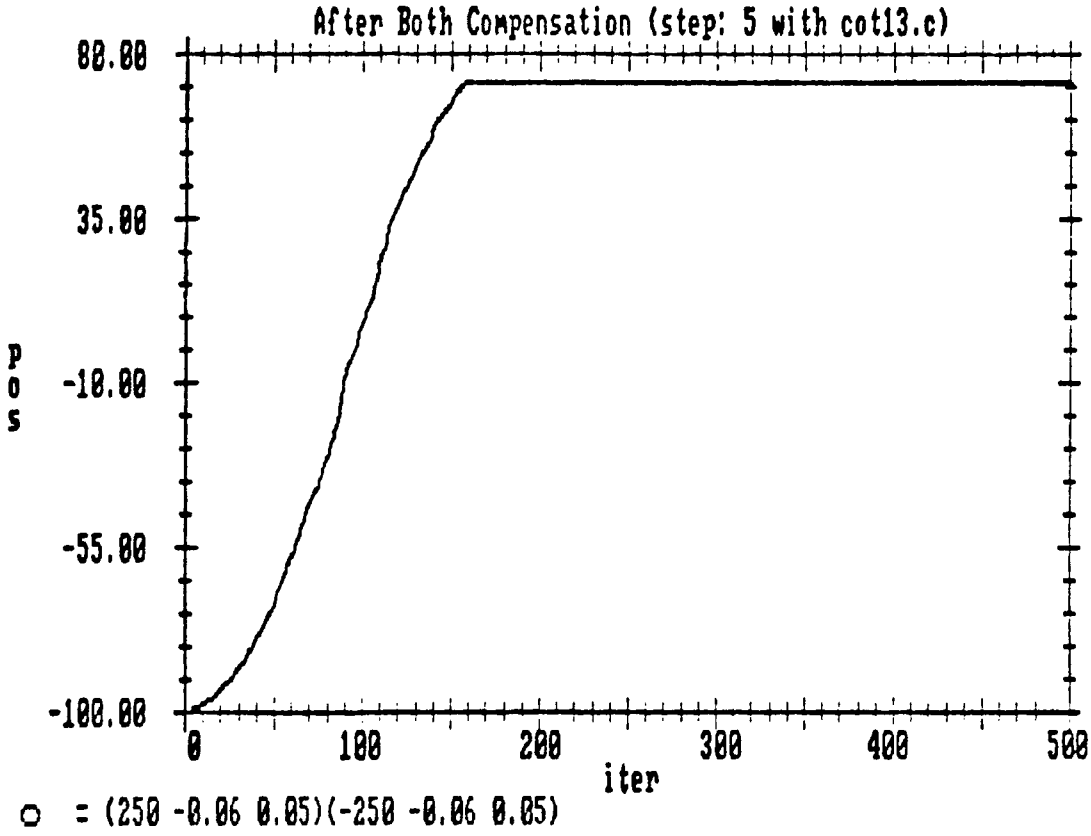


Figure 7-9 Hub Position Profile after Compensation for Both Characteristics

CHAPTER VIII

Conclusion

Our work demonstrates that the integration of the ideas in Chapter 4, 5, and 6 leads to striking improvement in controller performance. These ideas can be carried over to other situations to design good motion controllers. Our work also shows that the investigation of the characteristics of the actuator is important. Every drive system should include compensators for nonlinearities such as ripple torque and friction.

Although the geometrically exact mathematical model derived in Chapter 3 was not used in doing the design here, it paves the way in the future to analyze it and design a suitable controller for it. This is definitely a challenging task. In fact, it has been started. The idea of modeling the beam as a chain of rigid bodies has been suggested and tried. This can be regarded as an approximation of the exact beam model. The existence and uniqueness properties of the exact equation should also be studied.

Another interesting idea is to use piezoelectric material, such as PVF_2, to increase the beam damping by active control and make it more robust. Because of the distributed structure of the film, the exact beam equation should be used. This is also a next step to consider for experimentation.

References

[1] Walrath, Craig D. "An Adaptive Bearing Friction Compensator based on Recent Understandings of Dynamic Friction Behavior." *Automatica*, Vol. 20, No. 6, 1984, pp. 717–727.

[2] Åström, K. J., B. Wittenmark, *Computer Controlled System: Theory and Design.* Prentice-Hall, 1984.

[3] Canudas, C., K.J. Åström, , K. Braun, "Adaptive Friction Compensation in DC Motor Drives." *Proceedings IEEE Robotics and Automation Conf.*, 1986. pp. 1556–1561.

[4] Heiserman, D. L. *How to Design and Build Your Own Custom Robot.* Tab Books Inc., 1981.

[5] Uffenbeck, J. *Microcomputers and Microprocessors.* Prentice-Hall Inc., 1985.

[6] Metrabyte Corporation *DASH-16 Manual.* 1984.

[7] Metrabyte Corporation *DDA-06 Manual.* 1984.

[8] Kuo, B.C. *Automatic Control Systems. 3rd Ed.* Prentice-Hall, 1975.

[9] Kailath, T. *Linear Systems. 2nd Ed.* Prentice-Hall, 1980.

[10] Fan, M.K.H., L.-S. Wang, J. Koninckx, A.L. Tits, "CONSOLE: A CAD Tandem for Optimization-Based Design Interacting with Arbitrary Simulators" To be published.

[11] Frank, G.H. "Design and Real-time Control of a Flexible Arm" *Master's Thesis*, University of Maryland, 1986.

[12] Åström, K.J. "SIMNON Tutorial"

[13] Fan, M.K.H., W.T. Nye, A.L. Tits, "DELIGHT.MaryLin User's Guide - 2/1/85", SRC Technical Report, TR-85-9, 1985.

[14] Goldstein, H. *Classical Mechanics 2nd ed.* Addison-Wesley Publishing Company, Inc., 1980.

[15] Weinstock, R. *Calculus of Variations: with the applications to physics and engineering,* McGraw-Hill, 1952.

[16] Juang, J.-N., L.G. Horta, H.H. Robershaw, "A Slewing Control Experiment for Flexible Structures" *J. Guidance, Control and Dynamics* Vol. 9, No. 5, 1986, pp. 599–607.

[17] Vu-Quoc, Loc "Dynamics of Flexible Structures Performing Large Overall Motions: A Geometrically-Nonlinear Approach" Ph. D. Dissertation, Electronics Research Lab. UCB/ERL M86/36. University of California, Berkeley, 1986.

[18] Hanselmann, H. "Implementation of Digital Controllers-A Survey." *Automatica*, Vol. 23, No. 1, 1987, pp. 7–32.

[19] Cannon, R.H. Jr., E. Schmitz, "Initial Experiments on the End-Point Control of a Flexible One-Link Robot," *Inter. J. of Robotics Research*, Vol. 3, No. 3, 1984, pp. 62–75.

Appendix A

Microsoft C Program Listing of the Integral Controller for a Motor System

```
/*******************************************************/
/*    integral control of the motor system             */
/*******************************************************/
#include <process.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
extern int dash16();
extern int dda6();
extern void exit();

main( argc, argv )
int argc;
char *argv[];
{
   int mode, based16=784, d16io[5], flag, loadcnt0;
   int basedda6=768, pa, pb, dda6ofst=2005;
   int mot_inp, posref, noloop, noiter;
   int i, j;
   double position[1500], z[1500];
   double veldata, posdata, postole, posofst;
   double uf, cur_force, vforce;
   double velgain, d16ofst, D_A, A_D, velfkgn;
   double contfdbk, u_force, p, m, h, exphp, zy;

/*****   Initialization of dash16    *****/
   initd16();
/*****   Start Execution   *****/
   dda6( dda6ofst );
   printf(" Start execution\n");
   mot_inp = dda6ofst;
   dda6( mot_inp );
   printf(" Input step size ( 10 or less )   ");
   scanf("%d", &posref );
   printf( " Input no of iterations " );
   scanf("%d", &noiter );

/*****   Input initial position data   *****/
   pa = inp( basedda6+12 );
   pb = inp( basedda6+13 );
   posofst = ((pb * 256 + pa) >> 4) * 3.1415926 / 2048;
/*****   Initialization of the loop  *****/
   noloop = 1000;
   posdata = 0;
   veldata = 0;
   h = 0.005;
   p = 26.3678;
   m = 13.7526;
```

```
   exphp = exp( - h*p );
   xy = m * (1 - exphp) / p;

/*****   Start the loop   *****/
  for( i = 0; i < noiter; i++ ) {
   z[0] = 0;
   position[0] = 0;
   for( j = 1; j < noloop; j++ ) {
   /*** Load Counter 0           */
      mode = 11;
      di6io[0] = 500; /*   Load Counter 0 1000 */
      dash16( &mode, di6io , &flag );
      if ( flag > 0 ) error( flag, mode );
      cur_force = posref;
   /*****   Input velocity   *****/
      mode = 1;
      di6io[0] = 0;
      di6io[1] = 0;
      dash16( &mode, di6io, &flag );
      if( flag > 0 ) error( flag, mode );
      mode = 3;
      dash16( &mode, di6io, &flag );
      if( flag > 0 ) error( flag, mode );
      veldata = di6io[0] * 2.425 / 204.8;
   /*****   Input position data   *****/
      pa = inp( basedda6+12 );
      pb = inp( basedda6+13 );
      posdata = ((pb*256+pa) >> 4)*3.1415926/2048-posofst;
      position[j] = posdata;
   /*****   Computing the artificial state   *****/
      z[j] = exphp * z[j-1] + xy * position[j-1];
   /*****   Computing the feedback value   *****/
      contfdbk = 8.45158*posdata-0.140695*veldata-14.2896*z[j];
      u_force = cur_force - contfdbk;
      uf = u_force * 204.8;
      if( uf > 2040 )   uf = 2040;
      if( uf < -2040 )  uf = -2040;
      mot_inp = uf + 2048;
      dda6( mot_inp );

      mode = 12;
      di6io[0] = 1;    /*   Latch before read  */
      dash16( &mode, di6io, &flag );
      if ( flag > 0 ) error( flag, mode );
      while( di6io[1] > 10 ) {
         mode = 12;
         di6io[0] = 1;
         dash16( &mode, di6io, &flag );
         if ( flag > 0 ) error( flag, mode );
         }
      }
  posref = -posref;
  }
/*****   The End   *****/
   dda6( dda6ofst );
}
```

Appendix B

Input File for CONSOLE and SIMNON in the Speed Control Study

```
/*********************************************************************
   Problem Description File for CONSOLE
*********************************************************************/
design_parameter Kp  init=1.07312e+00
design_parameter Ti  init=2.03349e-01
/*   Meet the engineering specification   */
functional_objective "speed"
   for t from 0 to 1 by 0.001
   minimize {
      double Ytr();
      return Ytr("z",t); }
   good_curve = {
       if( t <= 0.6 ) return 1.1;
       else           return 1.01; }
   bad_curve = {
       if( t <= 0.6 ) return 1.2;
       else        return 1.05;  }
functional_objective "undershoot"
   for t from 0.4 to 1 by 0.001
   maximize {
      double Ytr();
      return Ytr("z",t); }
   good_curve = {
       if( t <= 0.5 ) return 0.90;
       else        return 0.99;   }
   bad_curve = {
       if( t <= 0.5 ) return 0.85;
       else        return 0.95; }

##########################################
#   System Description File for CONSOLE
##########################################
K = 6
Kt = 21.62
Kb = 2.443
Ra = 33.6
J = 0.1
subsystem plant = [ (0) K*Kt*Kp/(Ra*Ti);   \
                    (2) J, (K*Kt*Kb*Kp+Kt)/(Ra*Kb), K*Kt*Kp/(Ra*Ti) ]
external_input u=1
external_output z
#   Set up the plant
connection plant_input = u
connection z = plant_output

""""""""""""""""""""""""""""""""""""""""
"   Input File for SIMNON
""""""""""""""""""""""""""""""""""""""""
continuous system CSPEED
```

```
"   Continuous-Time linear system for the motor speed control
input u
output y
state x
der dx
dx=-Kt/(J*Ra*Kb)*x+K*Kt/(J*Ra)*u
y=x
yout=x
K:6
Kt:21.62
Kb:2.443
J:0.1
Ra:33.6
END

discrete system DSPEED
"Discrete-Time PI Feedback for the motor speed control
input yr y
output u
state i j
new ni nj
time t
tsamp ts
u=i
ni=i-Kp*(y-j)+(Kp*h/Ti)*(yr-j)
nj=y
ts=t+h
Kp:1.07312
Ti:2.03349e-1
h:0.002
END

continuous system CSPEED
"Continuous-Time linear system for the motor speed control
input u
output y
state x
der dx
dx=-Kt/(J*Ra*Kb)*x+K*Kt/(J*Ra)*u
y=x
yout=x
K:6
Kt:21.62
Kb:2.443
J:0.1
Ra:33.6
END
```

Appendix C

Input File to CONSOLE and SIMNON for Integral Control of the Flexible Arm

```
/*************************************************************
   Input file for CONSOLE: specification description file
 *************************************************************/
PI = 3.1415926
R1 = 1/(9*2*PI)
R2 = 1/(20.6*2*PI)
R3 = 1/(9.5*2*PI)
design_parameter k1 init=8.23015 vari=1.43e-1
design_parameter k2 init=-2.78527e-01 vari=0.9e-3
design_parameter k3 init=-2.23363e-04 vari=8.4e-5
design_parameter k4 init=-3.564e-5 vari=1.7e-5
design_parameter k5 init=-5.7879e1 vari=4.34e1
design_parameter k6 init=5.79548e-1 vari=0.5e-1
design_parameter k7 init=9.92761e-1 vari=1e-2
design_parameter m init=1.03611e2 vari=1e1
design_parameter p init=2.38868e1 vari=1e0 min=0.1
functional_objective "over-short"
   for t from 0 to 2 by .0005
   minimize {
      double Ytr();
      return Ytr(1,t)/0.15664;}
   good_curve = {
       if( t <= 0.8 ) return 1.06;
       else        return 1.005; }
   bad_curve = {
       if( t <= 0.8 ) return 1.1;
       else        return 1.01; }

functional_objective "settling"
   for t from 0.5 to 2 by .0005
   maximize {
      double Ytr();
      return Ytr(1,t)/0.15664; }
   good_curve = {
       if( t <= 0.6 ) return 0.9;
       else        return 0.999; }
   bad_curve = {
       if( t <= 0.6 ) return 0.85;
       else        return 0.99; }

objective "steadystate"
   minimize {
      double Ytr();
      return fabs( Ytr(1,100.0)-0.15664 ); }
   good_value = 0
   bad_value =  1

functional_constraint "max-accel" hard
   for t from 0 to 2 by .005
```

```
   {  double Ytr();
      return Ytr(2,t); }
   <=   good_curve = { return 10; }
        bad_curve = { return 11; }

functional_constraint "max-accel" hard
   for t from 0 to 2 by .005
   {  double Ytr();
      return Ytr(2,t); }
   >=   good_curve = { return -10; }
        bad_curve = { return -11; }

functional_constraint "control" hard
   for t from 0 to 2 by .002
   {  import k1 k2 k3 k4 k5 k6 k7
      double Ytr();
      return fabs( 1-(k1*Ytr(3,t)+k2*Ytr(4,t)+k3*Ytr(5,t)+k4*Ytr(6,t)
             +k5*Ytr(7,t)+k6*Ytr(8,t)+k7*Ytr(9,t)));  }
   <=   good_curve = {   return 5;  }
        bad_curve = {    return 6;  }

constraint "observ1" hard
   {   import m p R1
       return fabs( (-p-m)*(-p-m)*R1*R1+0.22*R1*(-p-m)+1 );
   }   >=   good_value=1
            bad_value =0.01

constraint "observ2" hard
   {   import m p R2
       return fabs( (-p-m)*(-p-m)*R2*R2+0.04*R2*(-p-m)+1 );
   }   >=   good_value=1
            bad_value =0.01

constraint "observ3" hard
   {   import m p R3
       return fabs( R3*(-p-m)+1 );
   }   >=   good_value=1
            bad_value =0.01

constraint "control1" hard
   {   import p R3
       return fabs( -p*p*p*R3*R3+2.1*R3*p*p-p );
   }   >=   good_value=1
            bad_value =0.01

######   Input file for CONSOLE: system description file #####
K=6.02
Kb=2.443
PI = 3.1415926
G = 0.125
R1 = 9*2*PI
R2 = 20.6*2*PI
R3 = 9.5*2*PI
R4 = Kb*G/(0.383*K)
A1 = 0.04*R2+0.22*R1
A2 = R1*R1+0.0088*R2*R1+R2*R2
A3 = 0.22*R2*R2*R1+0.04*R2*R1*R1
A4 = R2*R2*R1*R1
B2 = R1*R2*R1*R2/(R3*R3)
B3 = 2*B2*R3
B4 = B2*R3*R3
```

```
C1 = (A1*A1-A2)*0.383*R4
C2 = -A1*0.383*R4
C3 = 0.383*R4
C4 = B2*0.383*R4
b6 = 2*PI*1.16*K/Kb

system_size Ninputs=1 Nstates=7 Noutputs=9
readmatrix A
-A1       1       0       0       0       0         0
-A2   0   1   0   B2    0         0
-A3       0       0       1       B3      0           0
-A4       0       0       0       B4      0           0
0         0       0       0       0       1           0
-b6*k1    -b6*k2  -b6*k3  -b6*k4  -b6*k5  -7.29-b6*k6  -b6*k7
m*R4      0       0       0       0       0           -p

readmatrix  B
0
0
0
0
0
b6
0

readmatrix C
R4      0       0       0       0       0       0
C1      C2      C3      0       C4      0       0
1       0       0       0       0       0       0
0       1       0       0       0       0       0
0       0       1       0       0       0       0
0       0       0       1       0       0       0
0       0       0       0       1       0       0
0       0       0       0       0       1       0
0       0       0       0       0       0       1

readmatrix Ut
1

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"   Input file for SIMNON: Plant
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
continuous system CFXINT
"Continuous-Time linear system for the flexible arm
input u
output y1 y2 y3
state x1 x2 x3 x4 x5 x6
DER dx1 dx2 dx3 dx4 dx5 dx6
dx1=a11*x1+a12*x2
dx2=a21*x1+a23*x3+a25*x5
dx3=a31*x1+a34*x4+a35*x5
dx4=a41*x1+a45*x5
dx5=a56*x6
dx6=a66*x6+b6*u
y=c1*x1
y1=x5
y2=x6
y3=c31*x1+c32*x2+c33*x3+c35*x5
a11:-17.6180513008
a12:1
a21:-20015.22226558215
```

```
a23:1
a25:15035.98801734754
a31:-224975.7773142842
a34:1
a35:1795004.050181533
a41:-53572128.68969316
a45:53572128.68969316
a56:1
a66:-7.29
b6:17.96018783816619
c1:0.15365855
c31:-999.5616948178257
c32:-0.8937063813923256
c33:0.05072674418604651
c35:762.7267177404495
end

"""""""   Input File for SIMNON: Controller    """""""
discrete system DFXARM
"Discrete-Time controller-observer for the flexibla arm
INPUT u
OUTPUT v y1 y2 y3
state x1 x2 x3 x4 x5 x6
new nx1 nx2 nx3 nx4 nx5 nx6
time t
tsamp ts
nx1=a11*x1+a12*x2+a13*x3+a14*x4+a15*x5+a16*x6+b1*v
nx2=a21*x1+a22*x2+a23*x3+a24*x4+a25*x5+a26*x6+b2*v
nx3=a31*x1+a32*x2+a33*x3+a34*x4+a35*x5+a36*x6+b3*v
nx4=a41*x1+a42*x2+a43*x3+a44*x4+a45*x5+a46*x6+b4*v
nx5=a55*x5+a56*x6+b5*v
nx6=a66*x6+b6*v
v=uref-k1*x1-k2*x2-k3*x3-k4*x4-k5*x5-k6*x6   "Feedback law
y1=x5
y2=x6
y3=c31*x1+c32*x2+c33*x3+c35*x5
ts=t+h
a11:0.0423342040936935
a12:0.006288796726573081
a13:3.968318192470788e-05
a14:1.4419247560836916-07
a15:0.8756638574786695
a16:0.002833851583283285
a21:-142.524116824618
a22:0.1531305474427612
a23:0.006987937061501564
a24:4.222357235716548e-05
a25:188.9419331259212
a26:0.9049320220096725
a31:-3540.739460819261
a32:-16.6524525591923
a33:0.9473982538713215
a34:0.009873981509827604
a35:19598.46977632989
a36:94.05509866750798
a41:-336904.2275392942
a42:-2125.912528886956
a43:-7.724697859376991
a44:0.9798380681541853
a45:381166.9406393609
a46:2238.98115014823
```

```
a55:1
a56:0.009644198250074845
a66:0.9296937947569544
b1:0.0001085674552153253
b2:0.04578293172629913
b3:4.821345333692129
b4:124.2241265603102
b5:0.0007555668825541574
b6:0.1492995476578602
c31:-1159.653162308412
c32:-1.036843885404896
c33:0.05885122410546139
c35:884.8863004559524
k1:30.36213161279457
k2:-0.1644232615154454
k3:-0.001473703749971524
k4:-6.848483626764808e-06
k5:-27.60053359991405
k6:0.5208171671299313
h:0.01
uref:1
END

""" Connecting File: Interface between Controller and Plant """
CONNECTING SYSTEM fxintcon
y1dif=y1[cfxint]-y1[dfxint]
y2dif=y2[cfxint]-y2[dfxint]
y3dif=y3[cfxint]-y3[dfxint]
u[dfxint]=v[dfxint]+l1*y1dif+l2*y2dif+l3*y3dif
u[cfxint]=v[dfxint]
l1:10.15
l2:.75
l3:0.05
END
```

Appendix D

Microsoft C Program Listing of the Integral Controller-Observer for the Flexible Arm

```
#include <process.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
extern int dash16();
extern int dda6();
extern void exit();
extern int abs();
extern FILE *fopen();
extern int fprintf();

int mode, based16=784, d16io[5], flag;
int basedda6=768, pa, pb, dda6ofst=2005;
int zero=0, one=1, three=3, eleven=11, four=4, twelve=12;

main()
{
   FILE *fp1, *fp2, *fp3, *fp4, *fp5, *fp6;
   char *comma = ",";
   int mot_inp, nodemo, noloop;
   int i, j;
   int loopcount, loopofst;
   double pos_ref, vel, pos, accel, ou_force;
   double uf, cur_force, acceldif, posdif, veldif, accelst;
   double velofst, accelofst, posofst;
   double st1, st2, st3, st4, st5, st6;
   float posplot[600], velplot[600], accelplot[600], state5[600];
   float state6[600], stateaccel[600];
   double nextst1, nextst2, nextst3, nextst4, nextst5, nextst6;
   double z[1000], m, h, p, zy, exphp;
   double contfdbk=0, u_force, u_tmp, sum, accelgain, velstgain;
   double offset(), riptorq, ripgain, ripofst, time;
   double Tf, wpos, wneg, wof, alpha1, alpha2, gamma1, gamma2;
   double u_low, u_high;

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

/*****   Open Files   *****/
   if( (fp1 = fopen( "itglpos.plo", "w" )) == NULL ) {
      fprintf( stderr, "couldn't open file itgl2.plo\n" );
      exit( 1 );
      }

/*****   Compute the offset value for velocity and acceleration  *****/
   velofst = offset( 0, 2.425 );
   accelofst = offset( 4, 1.0 );
   printf("tach offset: %lf  accel offset: %lf\n", velofst,accelofst);
```

```
/*****   Start Execution   *****/
   printf(" Start execution\n");
   mot_inp = dda6ofst;
   dda6( mot_inp );

   time = 0.003;
   ripgain = 0.19 * 204.8;
   ripofst = 13.5;
   vpos = 245;
   vneg = -220; /* 1.31 * 204.8 */
   vof = 0.03 * 204.8;
   alpha1 = -0.1;
   gamma1 = 0.02;
   alpha2 = -0.05;
   gamma2 = 0.021;

   printf(" Input step size ( 5 or less)   ");
   scanf("%lf", &pos_ref );
   if( fabs( pos_ref ) > 5 ) pos_ref=5;
   printf(" How many demos  ");
   scanf("%d", &nodemo );
   printf( "No of Demos : %d\n", nodemo );

   u_high = 2000;
   u_low = -2000;
   cur_force = 0;
   outp( basedda6+15, 0xbe );
   m = 1.20263e1;
   p = 1.07294e1;
   h = 0.01;
   exphp = exp( -h*p );
   xy = m * (1 - exphp) / p;
   noloop = 500;
   loopcount = 1000;
   loopofst = 18;
/*****   Initialization of the loop  *****/
   st1 = st2 = st3 = st4 = st5 = st6 = 0;

/*****   Start the loop   *****/
   for( i = 0; i < nodemo; i++ )  {
      x[0] = 0;
      for( j = 0; j < noloop; j++ )  {
      /*   Load Counter 0 = loopcount */
         mode = eleven;
         d16io[0] = loopcount;
         dash16( &mode, d16io , &flag );
         if ( flag > 0 ) error( flag, mode );

      /*****   25 Hz Filter   *****/
         cur_force = 0.061 * pos_ref + 0.39 * cur_force;

      /*****   Input velocity   *****/
         mode = one;
         d16io[1] = d16io[0] = zero;
         dash16( &mode, d16io, &flag );
         if( flag > 0 ) error( flag, mode );
         mode = three;
         dash16( &mode, d16io, &flag );
         if( flag > 0 ) error( flag, mode );
         vel = d16io[0] * 2.425 - velofst;  /*  2.425/204.8  */
```

CURRICULUM VITAE

Name : Li-Sheng Wang.

Permanent adress : 4319 Rowalt Drive Apt. 201
College Park, Maryland 20740.

Degree and date to be conferred : Master of Science, 1987.

Date of birth : November 25, 1961.

Place of birth : Keelung, Taiwan, R.O.C.

Secondary education : Chenkuo Senior High School, 1979.
Taipei, Taiwan, R.O.C.

Collegiate institutions attended:	Dates	Degree	Date of Degree
National Taiwan University	Aug., 1979	B.S.	June, 1983.
University of Maryland	Aug., 1985	M.S.	Aug., 1987.

Major : Electrical Engineering.

```
     /*****   Input acceleration   *****/
        mode = one;
        di6io[1] = di6io[0] = four;
        dash16( &mode, di6io, &flag );
        if( flag > 0 ) error( flag, mode );
        mode = three;
        dash16( &mode, di6io, &flag );
        if( flag > 0 ) error( flag, mode );
accel = di6io[0] - accelofst;  /*  1/204.8  */

     /*****   Input position data   *****/
        pa = inp( basedda6+12 );
        pb = inp( basedda6+13 );
        pos = ((pb*256+pa) >> 4)*0.3141592653; /*PI/10*/

     /*****   Compute the ripple torque   *****/
        riptorq=ripgain*sin(0.2002*(pos-ripofst+time*vel));
        /*   0.2002 = 41/204.8   */

     /*****   Computing the feedback value   *****/
contfdbk = 1.83067645e1 * st1 - 1.08849454e-1 * st2
                  -2.28869e-3 * st3 + 1.471595e-5 * st4
                  +1.90587466e1 * st5 + 7.3769966e-1 * st6
                  -2.39013e-7 * z[j];
  u_tmp = cur_force * 204.8 - contfdbk;
     /***   compute the friction compensation feedback   ***/
        if     ( vel > wpos )                Tf = gamma1*vel+39;
        else if( wpos >=vel &&vel > wof )    Tf = alpha1*vel+59;
        else if( wof >=vel &&vel >= -wof ) {
                if( u_tmp > 2.5 )       Tf = 59;
                else if( u_tmp < -2.5 ) Tf = -40;
                else                    Tf = 0;
                }
        else if( -wof >vel &&vel >= wneg )   Tf = alpha2*vel-40;
        else                                 Tf = gamma2*vel-29;
           /* 59 = 0.288*204.8      40.14 = 0.196*204.8 */
        u_force = u_tmp - riptorq + Tf;
uf = -u_force;
if( uf > 2040 )   uf = 2040;
        if( uf < -2040 )  uf = -2040;
        mot_inp = uf + 2005;
        dda6( mot_inp );

posdif = pos + st5;
        posplot[j] = pos;
veldif = -vel + st6;
        accelst = 0.22 * (-9.99561695e2*st1-0.8937063814*st2
                   + 5.0726744186e-2*st3 + 7.6272671774e2 * st5);
acceldif = -accel - accelst;

     /* add the saturator to the input of observer */
        if( u_force < u_low )       u_force = u_low;
        else if( u_force > u_high ) u_force = u_high;
        else                        ;
        ou_force = u_force - (10.15 * posdif + 0.75 * veldif
                              + 0.05 * acceldif);

        nextst1 = 4.2334203834e-2 * st1 + 6.288796731e-3 * st2
                  + 3.968318193883e-5 * st3 + 1.44192475336e-7 * st4
                  + 8.75663857161e-1 * st5 + 2.833851578573e-3 * st6
```

```
                + 1.259558e-4 * ou_force;
        nextst2 = -1.425241169e2 * st1 + 1.53130547262e-1 * st2
                + 6.987937066e-3 * st3 + 4.22235723665e-5 * st4
                + 1.889419332e2 * st5 + 9.049320216434e-1 * st6
                + 5.31156024e-2 * ou_force;
        nextst3 = -3.5407394626e3 * st1 - 1.665245255e1 * st2
                + 9.47398254e-1 * st3 + 9.87398151e-3 * st4
                + 1.959846977627e4 * st5 + 9.405509866862e1 * st6
                + 5.5935400404 * ou_force;
        nextst4 = -3.3690422778e5 * st1 - 2.12591253e3 * st2
                - 7.724697845 * st3 + 9.798380682e-1 * st4
                + 3.8116694088453e5 * st5 + 2.238981149e3 * st6
                + 1.4412007e2 * ou_force;
 nextst5 = st5 + 9.64419825e-3*st6+8.7657973413e-4 * ou_force;
        nextst6 = 9.29693794757e-1 * st6 + 1.7321161212e-1 * ou_force;
        z[j+1] = exphp * z[j] + zy * st1 * 0.1324458;

 st1 = nextst1;
        st2 = nextst2;
        st3 = nextst3;
        st4 = nextst4;
        st5 = nextst5;
        st6 = nextst6;

        mode = twelve;
        d16io[0] = one;    /*   Latch before read  */
        dash16( &mode, d16io, &flag );
        if ( flag > 0 ) error( flag, mode );
        while( d16io[1] > loopofst ) {
        /* Reading Counter 0 */
           mode = twelve;
           d16io[0] = one;    /*   Latch before read  */
           dash16( &mode, d16io, &flag );
           if ( flag > 0 ) error( flag, mode );
           }
        }
     pos_ref = -pos_ref;
      }
/*****   The End   *****/
   dda6( 2005 );
   printf("End of loop\n");
   for( j = 0; j < noloop; j++ )
      fprintf( fp1,"%d%s%lf\n", j, comma, posplot[j]);
   printf("The Ends.\n");
}
```