LEARNING ALGORITHMS
FOR
MANIPULATOR CONTROL

By

CHARLES KEVIN HUSCROFT

B.A.Sc.(Hons), The University of British Columbia, 1979

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Department of Electrical Engineering)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1984

86

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date October 17, 1984

Abstract

A method of robot manipulator control is proposed whereby algorithms are used to learn sum of polynomials representations of manipulator dynamics and kinematics relationships. The learned relationships are utilized to control the manipulator using the technique of Resolved Acceleration Control. Such learning is achieved without recourse to analysis of the manipulator; hence the name Self-Learned Control.

Rates of convergence of several learning algorithms are compared when learning estimates of various non-linear, multi-variate functions. Interference Minimization is found to be superior to the Gradient Method, Learning Identification and the Klett Cerebellar Model. Simplification of the implementation of Interference Minimization is described. A variant, Pointwise Interference Minimization, is introduced that is suitable for certain applications.

Self-Learned Control with path specification in Cartesian coordinates is demonstrated for a simulated two link manipulator. It is shown that sum of polynomials representations of the inverse dynamics, inverse kinematics and direct position kinematics relationships are adequate to achieve control comparable to that achieved using their analytical counterparts and can be learned without analysis of the manipulator.

Further research is outlined to achieve automatic adaptation to tool mass, implementation of sum of polynomials estimators and enhancement of the Klett Cerebellar Model.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor, Dr. Peter D. Lawrence, for his enthusiastic encouragement, assistance and support. I would also like to thank fellow student, James J. Clark, for the many ideas he offered during our frequent conversations.

My wife, Dulce Estrella, has been a constant source of encouragement and my daughter, Alicia Estrella, has been a constant source of inspiration.

I dedicate this thesis to my nephew Lawrence Wyatt Telling, whom I will always remember.

This research has been supported by the Natural Sciences and Engineering Research Council of Canada through a Postgraduate Scholarship to its author and Grant #A4924, and by the University of British Columbia in the form of a Teaching Assistanceship.

1 INTRODUCTION

There is currently much interest in adaptive control in the field of robotics. Largely, this is due to the desire to avoid analysis of the dynamics and kinematics of robot manipulators. The goal of this thesis is to examine a family of learning algorithms and demonstrate their ability to learn the non-linear, multi-variate functions describing the dynamics and kinematics of a robot manipulator. Furthermore, it is intended to show that this learning can be done without recourse to analysis of the dynamics or kinematics of the manipulator.

The starting point for this work was the Cerebellar Model proposed by Klett [26]. The learning machine proposed by Klett represents an extension of previous cerebellar models such as those proposed by Albus [1,2] and Marr [31] and draws upon earlier work on perceptrons, and other learning machines [42,49]. The cerebellum is involved in maintaining posture and coordinating motor activities of the body [1,26,38]; i.e. manipulator control. Thus it is reasonable to consider application of the Klett Cerebellar Model in learned manipulator control.

Currently, most industrial robot manipulators are based on individual joint servo control. Path specifications in a task oriented coordinate system such as Cartesian coordinates must be transformed into path specifications in terms of successive joint positions, a time consuming task. Individual joint servo control typically neglects coupling dynamic effects and thus

performance is limited.

Manipulator control techniques have been proposed that make use of the complete analytical dynamics of a manipulator; an example is Resolved Acceleration Control as proposed by Luh et al [30]. The disadvantage of such techniques is that they require analysis of the dynamics and kinematics of each manipulator to be so controlled. This is often a difficult task. Furthermore, the parameters used in the formulation of the dynamics such as link lengths, masses, centers of gravity and moments of inertia may be difficult to obtain by anyone other than the manufacturer of the manipulator.

Adaptive techniques have been proposed to avoid the necessity of manipulator analysis and measurement of manipulator parameters [29]. One such technique is the Adaptive Linear Controller proposed by Koivo and Guo [27]. A locally valid model of the manipulator dynamics is estimated on-line, based on recent observations of applied torques and resulting manipulator motion. Performance is quite good and the method does not require *a priori* analysis of the manipulator or measurement of manipulator parameters. Unfortunately, many mathematical operations are necessary for the on-line operations. Intuitively, the Adaptive Linear Controller would seem to be inefficient because although it is continually adapting, i.e. learning, it is continually forgetting as well. The locally valid model is changed as the manipulator moves from place to place by incorporating information from new observations and forgetting information from old observations. Thus upon

repeating a path, the Adaptive Linear Controller does not benefit from the previous experience.

Self-Learned Control as proposed in this thesis differs from previous adaptive methods in that a globally valid model of the manipulator dynamics and kinematics is learned. In this respect it is somewhat similar to the Cerebellar Model Articulation Controller proposed by Albus[1,2]. Due to the use of continuous functions as the basis of our learned functional estimates, however, our method requires far fewer weighting coefficients than the method proposed by Albus and no special encoding of input variables. Once the dynamics and kinematics have been learned to a degree adequate for control purposes, no further learning need take place. This assumes, of course, that the manipulator dynamics and kinematics are not changing.

Interference Minimization, the principle learning algorithm used in this thesis, was derived by Klett as an intermediate step in the derivation of his Cerebellar Model [26]. Chapter 2 examines the convergence rates of Interference Minimization and related learning algorithms such as the Gradient Method [36,57] and Learning Identification [12,39,50]. A method of reducing the number of calculations required to implement Interference Minimization is introduced. Also introduced are Pointwise counterparts of several of these algorithms that have utility in certain applications.

In chapter 3 the aforementioned learning algorithms are applied to achieve Self-Learned control of a simulated two link manipulator. Section 3.1 discusses Resolved Acceleration Control

using path specification in Cartesian coordinates and section 3.2 provides the analytical dynamics and analytical kinematics of the two link manipulator. Sections 3.3 and 3.4 outline how the two link manipulator was simulated and how standard test path specifications were generated. In section 3.5 the analytical dynamics and analytical kinematics of the two link manipulator are used in a simulation of a realistic implementation of Resolved Acceleration Control that we call Standard Analytical Control. This establishes a performance benchmark for subsequent comparison with Self-Learned control. In sections 3.6 and 3.7 it is shown that the Cartesian inverse dynamics and direct position kinematics of the two link manipulator can be represented adequately as sums of polynomials over a workspace consisting of a sizable portion of the manipulator's reach. Furthermore, it is shown to be possible to Pre-Learn these sum of polynomials representations using the analytical Cartesian inverse dynamics and analytical direct position kinematics as a guide. Finally, in sections 3.8 and 3.9 it is shown that these representations can be Self-Learned without recourse to the analytical dynamics or analytical kinematics. In control using the Self-Learned Cartesian inverse dynamics as shown in section 3.8, a vision system is assumed to observe the manipulator and provide error correcting feedback information. Section 3.9 shows that utilization of the Self-Learned direct position kinematics permits replacement of the vision system once learning is complete.

The contributions of this thesis are summarized in chapter

4 and chapter 5 offers several detailed suggestions for areas of future research. First, there is the need to apply Self-Learning to a real manipulator having more than 2 degrees of freedom and to extend Self-Learning to allow the Cartesian inverse dynamics to be adjusted automatically to compensate for the attachment to the manipulator of a tool of any mass within a given range. Secondly, investigation is warranted into the implementation of the required sum of polynomials estimators. Finally, proposed modifications of the Klett Cerebellar Model may increase its plausibility.

2 INTERFERENCE MINIMIZATION AND RELATED LEARNING ALGORITHMS

2.1 LEARNED FUNCTIONAL ESTIMATION USING SUMS OF POLYNOMIALS

In adaptive control, it is often necessary to learn to estimate unknown functional relationships through a learning technique. In this thesis we study techniques whereby functional estimation is learned based on a sum of polynomials representation. One such technique is the Gradient Method used by Widrow et al [57] for adaptive filtering. Another technique is the method of Learning Identification that was formulated for linear systems by Nagumo and Noda [39]. Learning Identification was extended to non-linear systems by Roy and Sherman [50]. These and related methods have been investigated by other authors such as Klett [26], Eweda and Odile [14], Bitmead and Anderson [8], Chan and Babu [10,11], Johnson [25] and Billings [7]. These techniques have been used to estimate a function of time samples of a continuous signal [39,50,57]. They can also be generalized to estimate functions of arbitrary variables [26,10,11]. We are interested in the learning of non-linear functions of several variables for application in manipulator control.

This thesis describes an improved learning algorithm of which the previously mentioned algorithms are simplifications. They thus form a family of learning algorithms. The convergence rates of the various algorithms are compared and factors simplifying the implementation of the improved learning algorithm are discussed.

2.2 AN IMPROVED LEARNING ALGORITHM - INTERFERENCE MINIMIZATION

This thesis is concerned with algorithms that learn to estimate a function as a sum of polynomials, specifically the Kolmogorov-Gabor [18,50] polynomials. The estimate of a function is formed as a weighted sum of polynomial terms,

$$f = \sum_k w_k p_k(\bar{z}) = \bar{w}^T \bar{p} \tag{2.1}$$

The basis polynomial terms are the set,

$$\{p_k(\bar{z})\} = \{ \prod_{i=0}^{v} z_i^{e_i} \} \tag{2.2}$$

where,

$$\sum_{i=0}^{v} e_i = s \ , \quad e_i \text{ an integer} \tag{2.3}$$

$$z_0 = 1 \tag{2.4}$$

For example, if s=2 and v=2 then the polynomial terms are,

$$\{p_k(\bar{z})\} = \{ 1, z_1, z_2, z_1^2, z_2^2, z_1 z_2 \} \tag{2.5}$$

The space spanned by this set is given by v, the number of input variables, and s, the system order. The number of terms, m, in such a polynomial is given by,

$$m = \frac{(s+v)!}{s!v!} \tag{2.6}$$

An estimate formed in this manner can mimic multivariate, non-linear functions. Learning takes place by iteratively adjusting the weight vector at a series of training points until the estimate corresponds to the target function throughout the space.

An improved learning algorithm can be derived as follows: At each training point the weights are adjusted to eliminate the

error in the estimate while minimizing the change in the estimate at other points in the space. We call the algorithm, Interference Minimization. This strategy can be enforced by minimizing the function,

$$c = \int_S (\Delta\bar{w}^T\bar{p})^2 \, \delta S + a(\Delta\bar{w}^T\bar{p} - \Delta f) \qquad (2.7)$$

using the Lagrange multiplier technique, where S is the domain of the input variables, $\Delta\bar{w}$ is the change in the weight vector and Δf is the error in the estimation. The error in the estimate is given by,

$$\Delta f = \overset{*}{f} - f \qquad (2.8)$$

where $\overset{*}{f}$ is the target function. Setting the partial derivatives of c with respect to $\Delta\bar{w}$ and a to zero and solving yields the weight adjustment formula,

$$\Delta\bar{w} = \frac{\Delta f \mathbf{P}^{-1}\bar{p}}{\bar{p}^T\mathbf{P}^{-1}\bar{p}} \qquad (2.9)$$

where,

$$\mathbf{P} = \int_S \bar{p}\bar{p}^T \, \delta S \qquad (2.10)$$

The matrix **P** is real, symmetric and positive definite and thus the inverse, $\mathbf{P}^{-1}$ exists.

Interference Minimization was derived by Klett [26] as an intermediate step in the derivation of his Cerebellar Model and can be generalized to allow use of other basis sets. In this work we consider in depth the specific case where K-G polynomials form the basis set. This elicits comparison with other previously described learning algorithms based on K-G polynomials and permits simplification of the implementation of

Interference Minimization as is shown in section 2.6.

2.3 RELATIONSHIP TO SIMILAR LEARNING ALGORITHMS

Several learning algorithms previously described in the literature can be considered to be simplifications of Interference Minimization. In (2.9) it can be seen that the matrix P^{-1} does not change the magnitude of the weight adjustment vector $\Delta\vec{w}$, it only changes the direction. If the matrix P^{-1} is deleted from the weight adjustment formula, the estimation error at a training point is still eliminated, however, the effect of the adjustment at other points in the space is not minimized. This sub-optimal learning algorithm is Learning Identification, which was formulated by Nagumo and Noda [39] using first order polynomials, and by Roy and Sherman [50] using K-G polynomials,

$$\Delta\vec{w} = \frac{\Delta f\vec{p}}{\vec{p}^T\vec{p}} \qquad (2.11)$$

In (2.11) the denominator serves as a scaling variable that adjusts the step size. When changing the weights, the error in the estimate is exactly eliminated regardless of the location of the training point. If the denominator is replaced by a positive constant, then the direction of the weight adjustment vector is not changed; one still moves towards a new weight set that eliminates the estimation error at the training point. The adjustment formula becomes,

$$\Delta\vec{w} = u\Delta f\vec{p} \qquad (2.12)$$

This algorithm is known as the Gradient Method and has been

discussed by authors such as Widrow et al [57] and Eweda and Odile [14]. The gain factor u must be chosen such that adjustments are not so large as to cause divergence at any training points within the space. This can be ensured by choosing u such that,

$$u \leq 2 \min_{S}\{1/(\bar{p}^T\bar{p})\} \tag{2.13}$$

This algorithm is even less optimal than (2.11) since many iterations are required at most training points in the space just to eliminate the estimation error at the training point.

Two variants of Interference Minimization follow if one selects basis functions such that $\mathbf{P}^{-1}$ is equal to the identity matrix. One method is to form the estimate as,

$$f = \bar{w}^T\mathbf{Q}\bar{p} = \bar{w}^T\bar{q} \tag{2.14}$$

where,

$$\mathbf{Q} = [\int_S \bar{p}\bar{p}^T \, \delta S]^{-1/2} = \mathbf{P}(\bar{p})^{-1/2} \tag{2.15}$$

The matrix $\mathbf{Q}$ is real, symmetric and positive definite. The terms of the vector $\bar{q}$ are a set of orthonormal polynomials. With such a basis vector, the matrix $\mathbf{P}$ becomes,

$$\mathbf{P}(\bar{q}) = \int_S \bar{q}\bar{q}^T \, \delta S = \int_S \mathbf{Q}\bar{p}\bar{p}^T\mathbf{Q}^T \, \delta S = \mathbf{P}^{-1/2}\mathbf{P}\mathbf{P}^{-1/2} = \mathbf{I} \tag{2.16}$$

The weight adjustment formula now becomes,

$$\Delta\bar{w} = \frac{\Delta f \bar{q}}{\bar{q}^T\bar{q}} \tag{2.17}$$

This method is equivalent to Interference Minimization as given previously in (2.9); the weight adjustment eliminates the estimation error at the training point while minimizing the

change in the estimate at other points in the space. It really only represents a simplification of the weight adjustment formula at the expense of a more complex estimation formula involving orthonormal polynomials. We call this method, Orthogonal Interference Minimization.

As in (2.11), the denominator in (2.17) serves as a scaling variable that adjusts the step size, and can be replaced with a positive constant;

$$\Delta\bar{w} = u\Delta f\bar{q} \tag{2.18}$$

To avoid divergence at any training point within the space, one must choose u such that,

$$u \leq 2 \min_{S}\{1/(\bar{q}^T\bar{q})\} \tag{2.19}$$

Despite its similarity to the Gradient Method, this algorithm is superior due to the use of orthonormal polynomial terms in $\bar{q}$. As with the Gradient Method, several iterations are required at a point if one is to eliminate the estimation error, however, each adjustment causes a minimal change in the estimate at other points in the space, thus speeding convergence. We call this algorithm the Cerebellar Model as it was originally proposed by Klett [26] as a model of learning in the mammalian cerebellum.

Discrete counterparts of several of the algorithms can be derived as follows: At each training point the weights are adjusted to eliminate error in the estimate at the point while minimizing the change in the estimate at previous training points, rather than minimizing the change in the estimate at all other points in the space as was done for (2.7). The Lagrange

multiplier technique can be used to obtain a weight adjustment formula identical to that of Interference Minimization, except that in place of the **P** matrix there is a matrix $\mathbf{D}_n$ which is used at the nth training point;

$$\mathbf{D}_n = 1/n \sum_{i=0}^{n-1} \bar{p}_i \bar{p}_i^T = [(n-1)\mathbf{D}_{n-1} + \bar{p}_n \bar{p}_n^T]/n \qquad (2.20)$$

where,

$$\mathbf{D}_0 = \mathbf{I} \qquad (2.21)$$

We call this algorithm, Pointwise Interference Minimization. As with Interference Minimization, two variants can be implemented in which polynomial terms in the basis vector are made orthogonal using the matrix,

$$\mathbf{E}_n = \mathbf{D}_n^{-1/2} \qquad (2.22)$$

We call these variants, Pointwise Orthogonal Interference Minimization and the Pointwise Cerebellar Model. The various algorithms are summarized in table 2.1.

METHOD	NAME	ESTIMATE	WEIGHT ADJUSTMENT	NOTES
1	Gradient Method	$f=\bar{w}^T\bar{p}$	$\Delta\bar{w}=u\Delta f\bar{p}$	
2	Learning Identification	$f=\bar{w}^T\bar{p}$	$\Delta\bar{w}=\frac{\Delta f\bar{p}}{\bar{p}^T\bar{p}}$	
3	Interference Minimization	$f=\bar{w}^T\bar{p}$	$\Delta\bar{w}=\frac{\Delta f\mathbf{P}^{-1}\bar{p}}{\bar{p}^T\mathbf{P}^{-1}\bar{p}}$	$\mathbf{P}=\int_S \bar{p}\bar{p}^T\,dS$
4	Orthogonal Interference Minimization	$f=\bar{w}^T\bar{q}$	$\Delta\bar{w}=\frac{\Delta f\bar{q}}{\bar{q}^T\bar{q}}$	$\bar{q}=\mathbf{Q}\bar{p}$, $\mathbf{Q}=\mathbf{P}^{-1/2}$
5	Cerebellar Model	$f=\bar{w}^T\bar{q}$	$\Delta\bar{w}=u\Delta f\bar{q}$	$\bar{q}=\mathbf{Q}\bar{p}$
6	Pointwise Interference Minimization	$f=\bar{w}^T\bar{p}$	$\Delta\bar{w}=\frac{\Delta f\mathbf{D}_n^{-1}\bar{p}}{\bar{p}^T\mathbf{D}_n^{-1}\bar{p}}$	$\mathbf{D}_n=1/n\sum_{i=0}^{n-1}\bar{p}_i\bar{p}_i^T$
7	Pointwise Orthog. Interference Minimization	$f=\bar{w}^T\bar{q}$	$\Delta\bar{w}=\frac{\Delta f\bar{q}}{\bar{q}^T\bar{q}}$	$\bar{q}=\mathbf{E}_n\bar{p}$, $\mathbf{E}_n=\mathbf{D}_n^{-1/2}$
8	Pointwise Cerebellar Model	$f=\bar{w}^T\bar{q}$	$\Delta\bar{w}=u\Delta f\bar{q}$	$\bar{q}=\mathbf{E}_n\bar{p}$

Table 2.1 Summary of learning algorithms

2.4 CHOICE OF TEST CONDITIONS FOR COMPARISON OF ALGORITHMS

In comparing the various learning algorithms, a convenient first step is to normalize all input variables. Comparison under such conditions allows one to see characteristics that are fundamental properties of the algorithms rather than just anomalies occurring due to particular combinations of input variable ranges. In practice, input variables are normally confined to a finite range of values and thus there is not a

loss of generality in normalizing all input variables. In order to allow positive and negative quantities to be represented, all inputs were normalized such that,

$$-1 \leq z_i \leq 1 \qquad (2.23)$$

Comparisons were made for various combinations of system order, s, and number of input variables, v. In all cases the function whose estimate was to be learned was chosen to be the K-G polynomial corresponding to s and v with all coefficients set to 1. For example, in the case s=2 and v=2, the function whose estimate was to be learned was,

$$\overset{*}{f} = 1 + z_1 + z_2 + z_1^2 + z_2^2 + z_1 z_2 \qquad (2.24)$$

We have found that relative rates of convergence when estimating these functions are representative of results when estimating functions described by K-G polynomials with coefficients randomly generated from a uniform distribution between -1 and 1. Figure 2.1 compares, for each of the learning algorithms, the results of five trials where the coefficients of the K-G target function were all 1's with the results of five trials where the coefficients were randomly chosen. All of the trials shown are for s=3 and v=3, but several other cases showed similar results.

The stopping rule for the various learning sequences was as follows: when the error, Δf, at a training point was found to be less than 0.01 at 100 successive training points then it was assumed that the error was less than 0.01 throughout the space and convergence was deemed to have occurred at the first of the 100 such training points. 100 points has proven to be an adequate test for convergence in the cases considered. Several

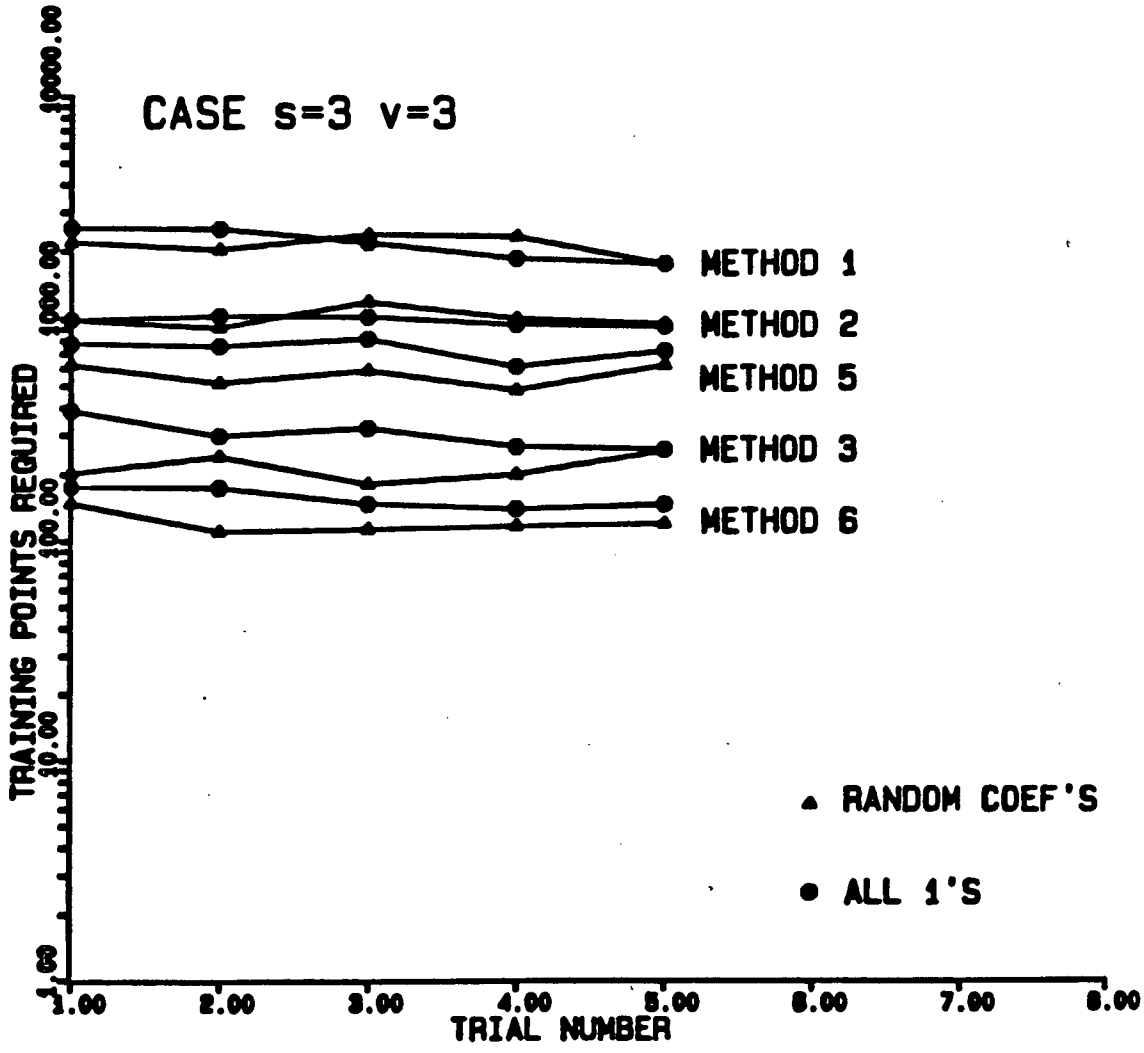


Figure 2.1 Convergence rates when target function is a polynomial with randomly chosen coefficients versus convergence rates when target function is a polynomial with 1's as coefficients, for case s=3 and v=3

trials were duplicated using a similar stopping rule where acceptance was required at more than 100 successive training points. Convergence occurred after the same number of points as before. The training points were successively chosen from uniform distributions of the input variables and one weight adjustment was performed at each point.

For those learning algorithms that require specification of the step size factor u, u was chosen to be as large as allowable without causing divergence to be possible. For Method 1, the Gradient Method, the condition for non-divergence given in (2.13) was satisfied by choosing,

$$\overset{*}{u} = 2/m \ . \qquad (2.25)$$

For Method 5, the Cerebellar Model, such an explicit formula for $\overset{*}{u}$ was not obtained; for each combination of s and v, a corresponding $\overset{*}{u}$ was obtained through simulations where,

$$\overset{*}{u} = 2 \min_{S}\{1/(\bar{q}^T\bar{q})\} = 2 \min_{S}\{1/(\bar{p}^T\mathcal{Q}^T\mathcal{Q}\bar{p})\} \qquad (2.26)$$

In all cases $\overset{*}{u}$ was obtained by considering 2000 randomly generated input variable vectors $\{z_1,..,z_v\}$. The $\overset{*}{u}$ factors so obtained are tabulated in table 2.2. It was verified in several cases that such a choice of u is near optimal for Method 5. Figure 2.2 shows the number of steps required for convergence as a function of u for the cases, s=1 and v=3, s=3 and v=3, s=3 and v=1. The choice of u given by (2.26) was near optimal in all cases except for s=3 and v=3. It was noted for this and other cases with a large number of terms, m, that it was typically possible to speed convergence by increasing u slightly from that

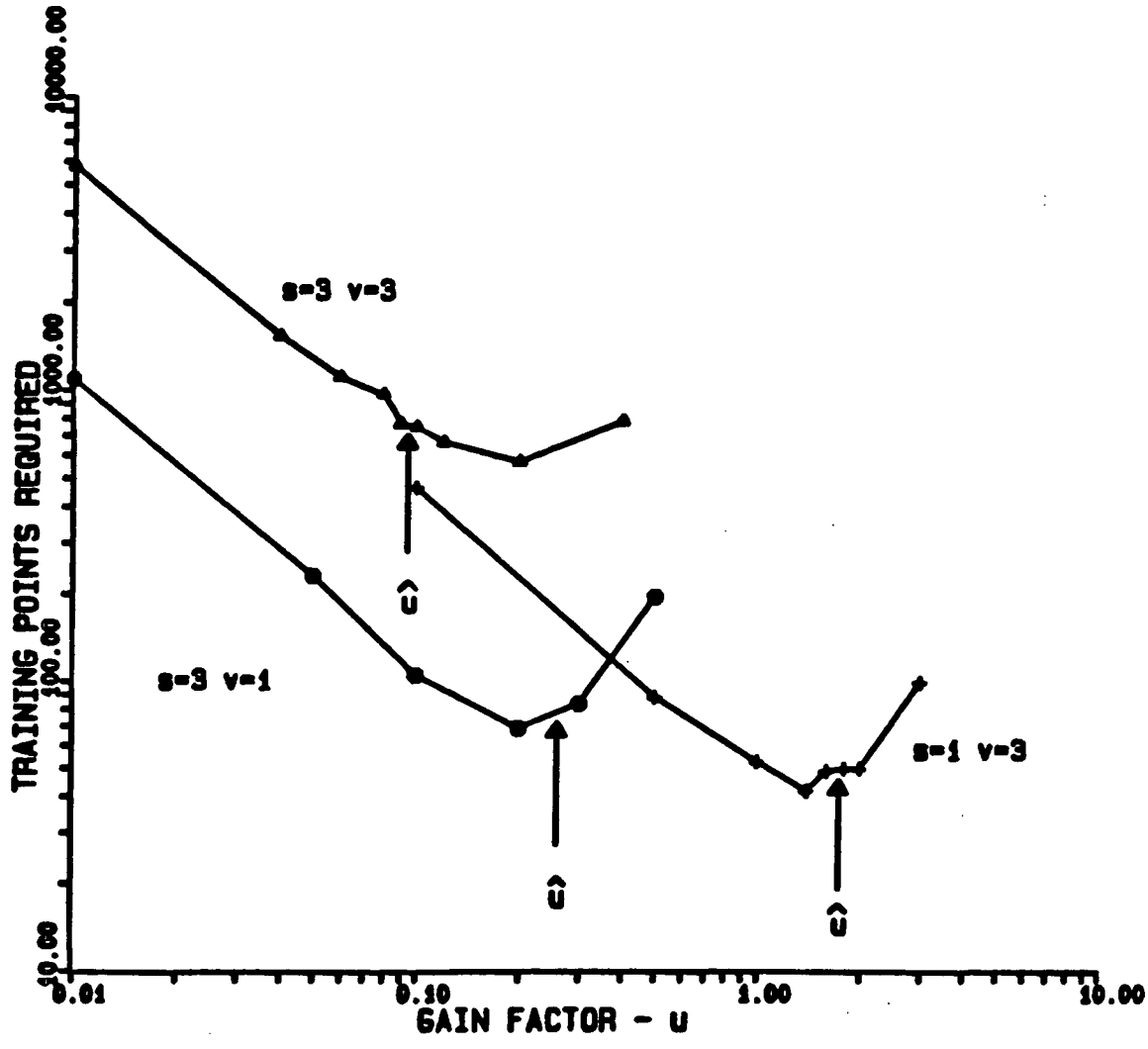


Figure 2.2 Convergence rates for Method 5, the Cerebellar Model, as a function of u for the cases s=1 and v=3, s=3 and v=3, and s=3 and v=1

given by (2.26). This entails a degree of risk as with such a choice of u the algorithm was divergent at some points in the space. Thus if occasional groups of several successive training points were in divergent regions of the space, the total algorithm would diverge. This was confirmed by several trials involving 10 or more adjustments per training point in which use of a u greater than that specified by (2.26) would lead to divergence. Thus it seems reasonable to consider $\overset{*}{u}$ as given by (2.26) to be the best choice when using Method 5, the Cerebellar Model.

System Order									
	6	.082	.022	----	----	----	----	----	----
	5	.112	.036	.015	----	----	----	----	----
	4	.161	.064	.034	----	----	----	----	----
	3	.251	.131	.093	.109	.147	----	----	----
	2	.455	.333	.327	.448	.676	1.08	1.68	3.00
	1	1.0	1.18	1.64	2.69	4.65	8.31	14.7	27.7
		1	2	3	4	5	6	7	8

Number of Variables

Table 2.2 Step size factors, $\overset{*}{u}$, for Method 5, the Cerebellar Model

2.5 COMPARISON OF THE LEARNING ALGORITHMS

Comparisons were made of all of the learning algorithms in table 2.1 except methods 4, 7 and 8. Methods 4 and 7 are

equivalent to methods 3 and 6, respectively, and converge in the same number of steps. Method 8 was included in table 2.1 for completeness only; in actual application, method 8 appears to require too much computation to be practical. The choice of u to ensure non-divergence depends on E_n which is changing as learning takes place. Thus the optimal u cannot be calculated beforehand for method 8.

Figures 2.3 through 2.8 show the number of steps required for the various learning algorithms to converge. The number of input variables, v, ranged from 1 to 8 and the system order, s, ranged from 1 to 6 in the cases investigated. The data represent the results for single trials except for the first order cases where the results represent the average of three trials.

Figure 2.9 shows the reduction of estimation error, Δf, as a function of the number of iterations for the case s=3 and v=3 using the various learning algorithms. The graph shows the magnitude of Δf, averaged over intervals of 100 training iterations.

For cases having a low system order there is not much difference between most of the various algorithms; the differences become pronounced only when higher order systems are considered. Method 1, the Gradient Method, is consistently the poorest algorithm. Method 3, Interference Minimization, is consistently much better than the Gradient Method; for 3rd, 4th, 5th and 6th order systems the factor of improvement is approximately 9, 20, 24 and 28, respectively. Method 2, Learning Identification, is similar in performance to the Gradient Method

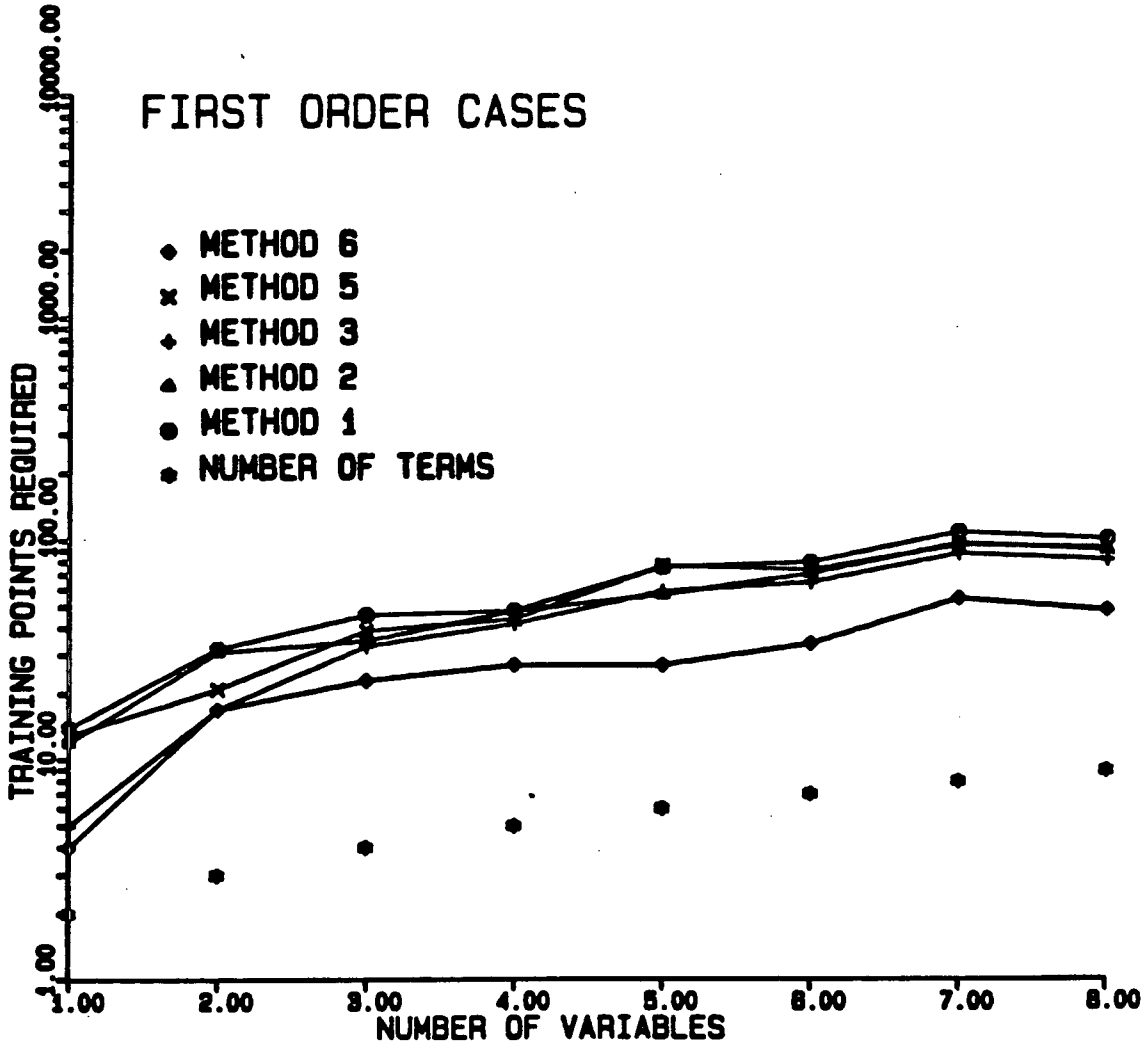


Figure 2.3 Convergence rates for various methods when s=1

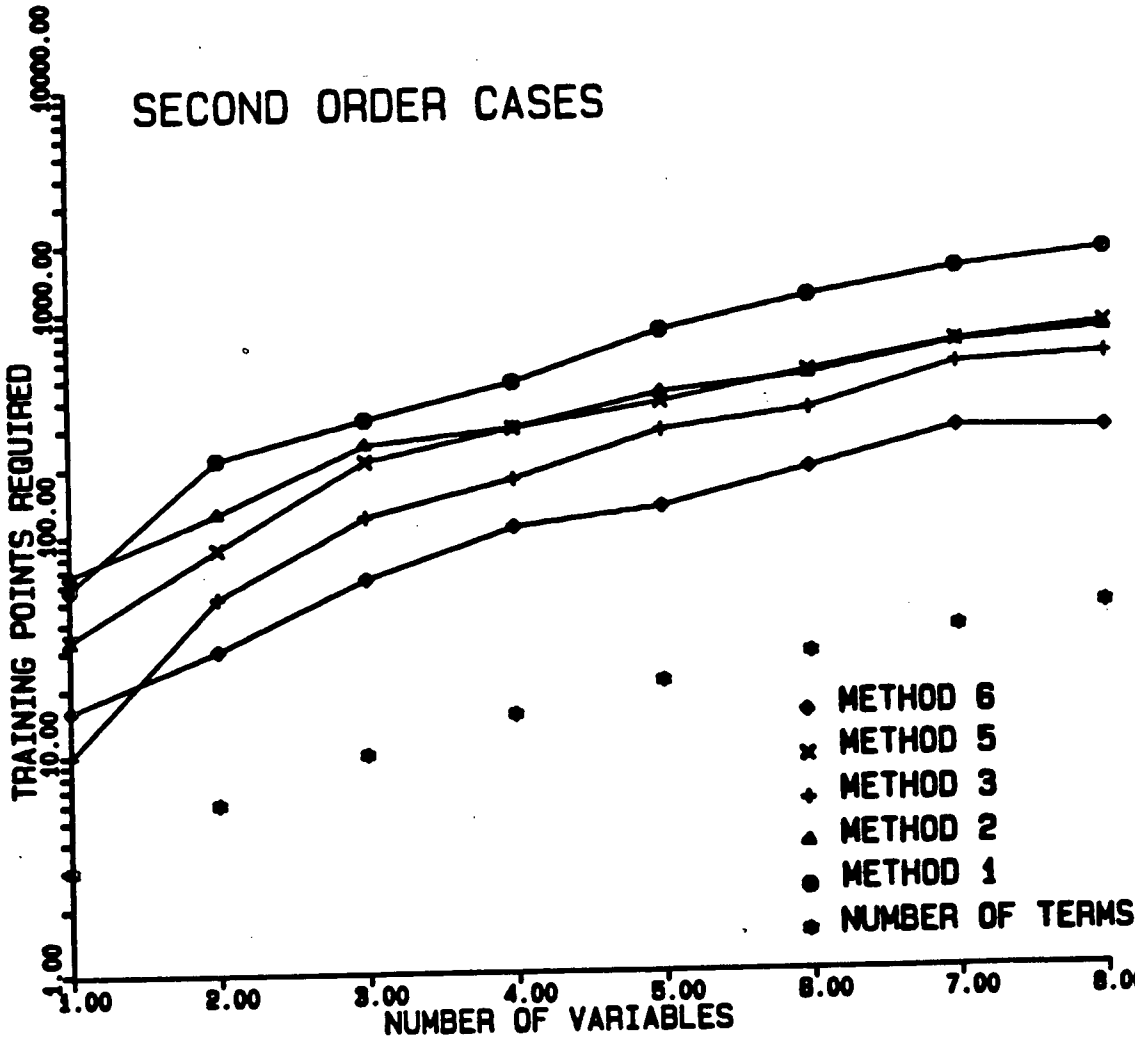


Figure 2.4 Convergence rates for various methods when s=2

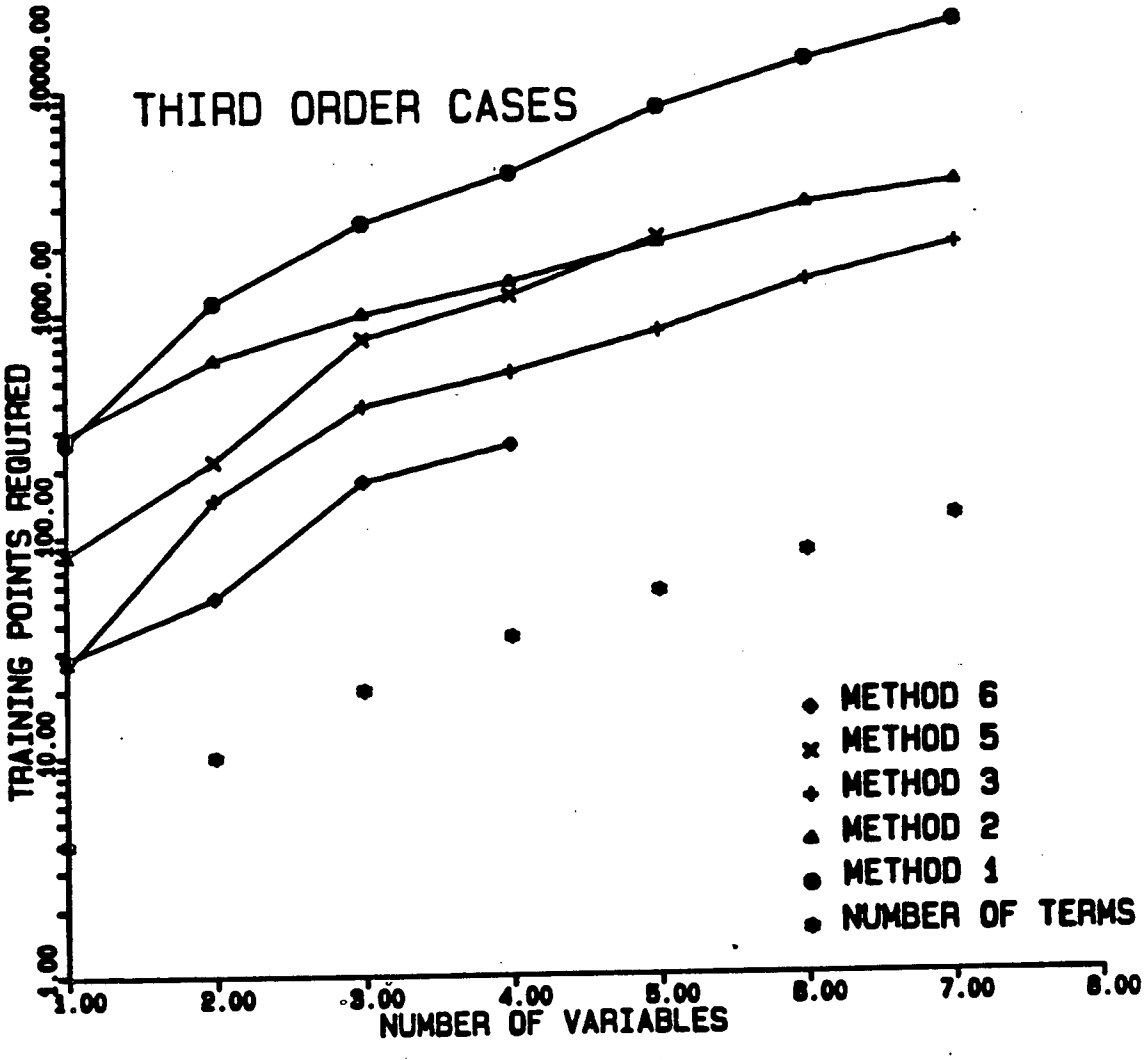


Figure 2.5 Convergence rates for various methods when s=3

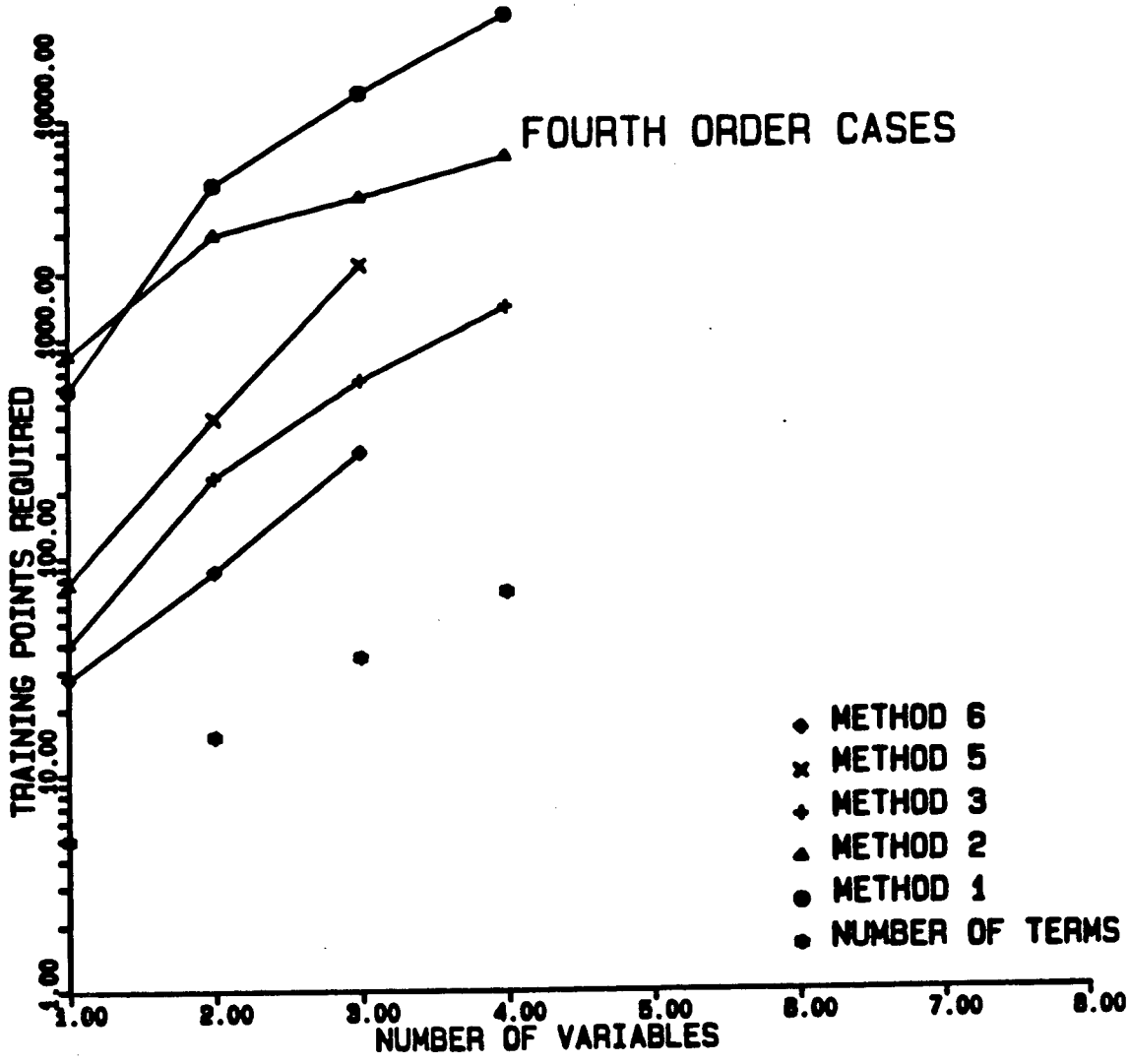


Figure 2.6 Convergence rates for various methods when s=4

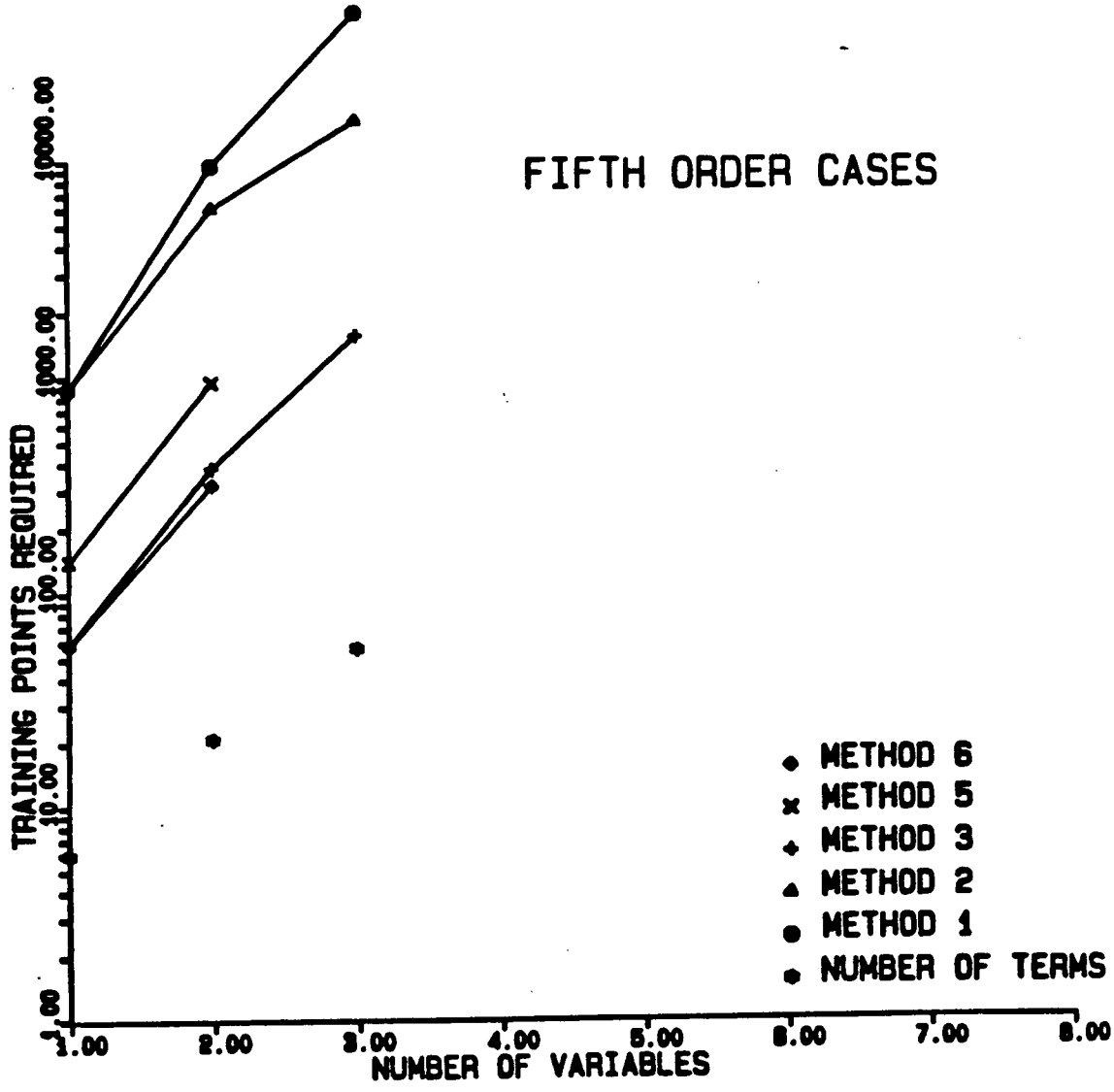


Figure 2.7 Convergence rates for various methods when s=5

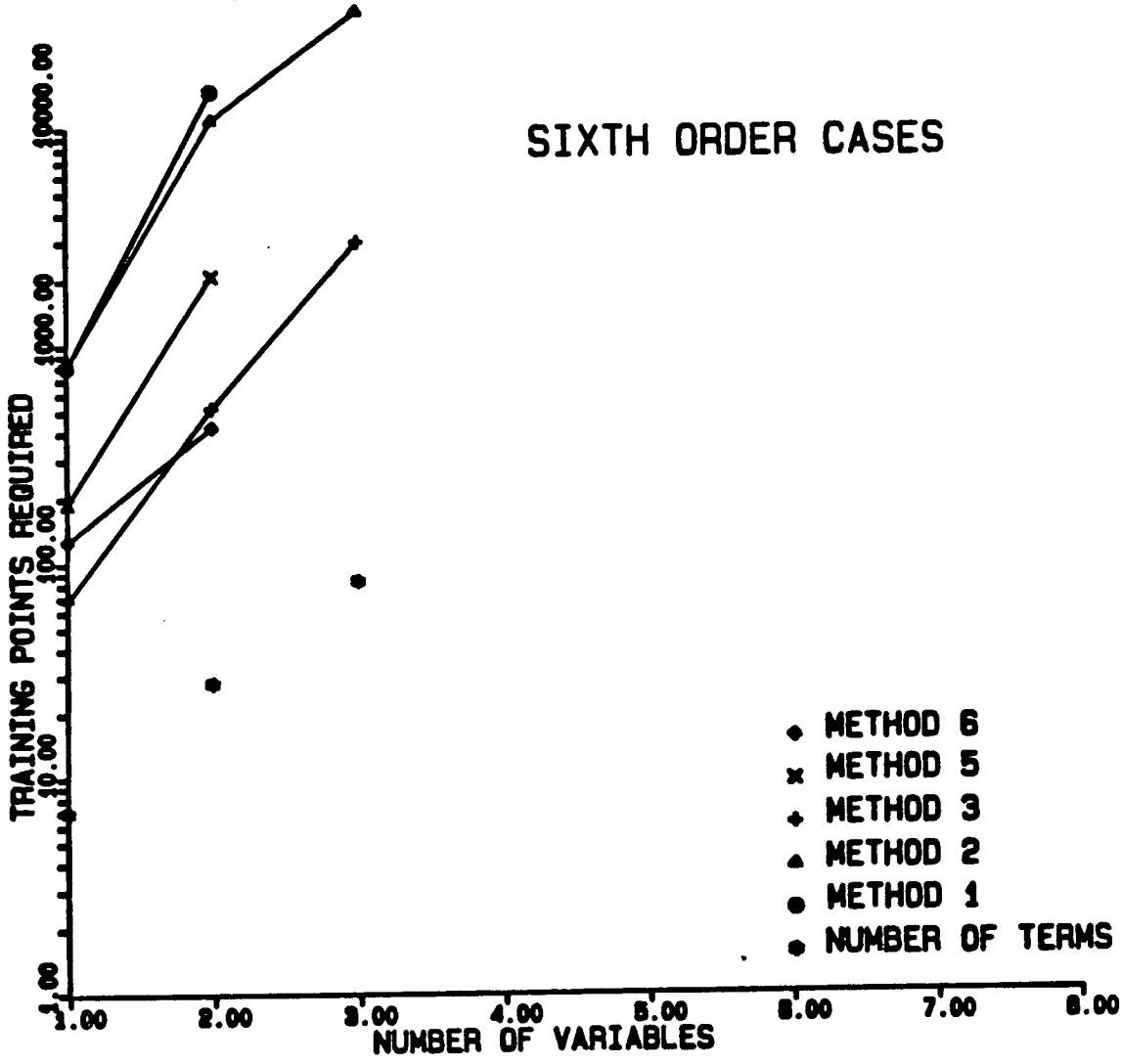


Figure 2.8 Convergence rates for various methods when s=6

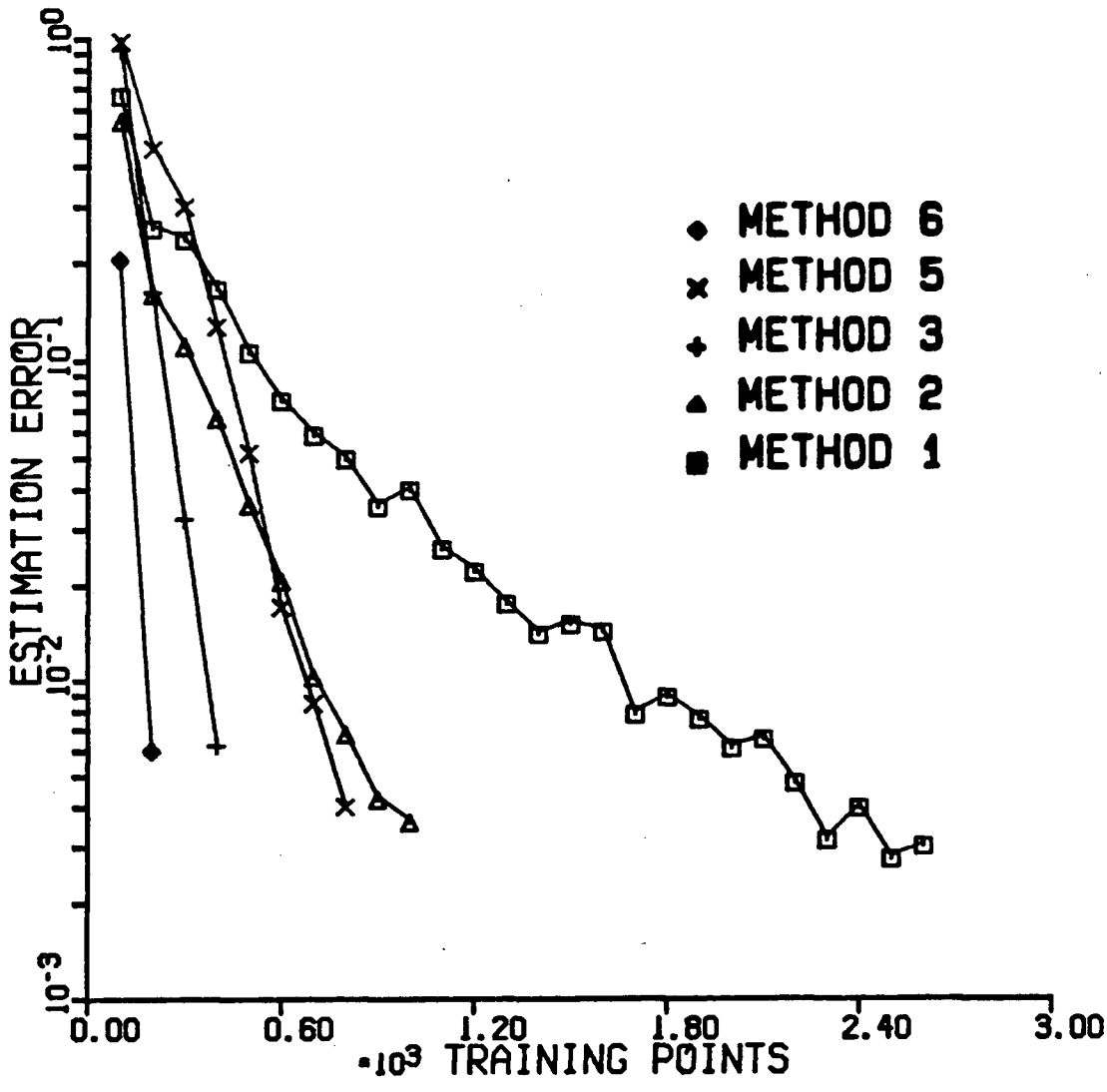


Figure 2.9 Reduction of estimation error, Δf, as a function of the number of training points for the case s=3 and v=3 using the various methods (estimation error averaged over each 100 iteration interval)

in cases where the system order, s, is larger than the number of input variables, v, however, Learning Identification is much better when v>s, approaching Method 3 in performance. Method 5, the Cerebellar Model, is intermediate in performance to the Gradient Method and Interference Minimization, much like Learning Identification; except that when s>v the Cerebellum Model is better than Learning Identification. Method 6, Pointwise Interference Minimization, is consistently the best method; for cases where v=1 the Pointwise Interference Minimization is similar to Interference Minimization and for cases where v>1 it is better by a factor of approximately 2, even when s=1.

The choice of which algorithm to use depends on the complexity of implementation as well as the achievable performance. Method 3, Interference Minimization, offers performance second only to Method 6, Pointwise Interference Minimization. As will be shown in section 2.6, the complexity of Interference Minimization is not as great as it might seem, thus simplifying implementation. Pointwise Interference Minimization, however, entails much more computation than the other algorithms as the matrix $\mathbf{D}_n$ must be updated and inverted yielding $\mathbf{D}^{-1}$ at each training point. Pointwise Interference Minimization is better than Interference Minimization by only a factor of 2 and thus typically does not warrant consideration. There are, however, certain situations in which the use of Pointwise Interference Minimization is appropriate. This is shown in section 2.7.

2.6 IMPLEMENTATION CONSIDERATIONS FOR INTERFERENCE MINIMIZATION

Interference Minimization achieves its improved performance over the Gradient Method and Learning Identification at the expense of a more complex algorithm. The increase in complexity, however, is not as great as it appears at first. If the input variables are normalized as in (2.23), then the matrix $\mathbf{P}^{-1}$ in (2.9) becomes sparse. $\mathbf{P}^{-1}$ has a special form that allows its sparseness to be easily utilized to speed up the computations required in Interference Minimization.

Consider the elements of the matrix **P**. The element P_{ij} is given by the integral,

$$P_{ij} = \int_S p_i(\bar{z})p_j(\bar{z})\ \delta S \qquad (2.27)$$

From the definition of $p_k(\bar{z})$ given in (2.2), (2.3) and (2.4) we have,

$$P_{ij} = \int_{-1}^{1}\cdots\int_{-1}^{1} \prod_{k=0}^{v} z_k^{(e_{ik}+e_{jk})}\ \delta z_1 \ldots \delta z_v \qquad (2.28)$$

If it occurs that $(e_{ik}+e_{jk})\text{mod}2=1$ for any variable, z_k, then $P_{ij}=0$. At least half of the elements P_{ij} are zero and for v>>s the vast majority of the elements P_{ij} are zero. An examination of the nonzero elements in **P** reveals a significant pattern. If the polynomials in the set $\{p_k(\bar{z})\}$ are ordered appropriately, the special banded nature of **P** will become apparent. The matrix has a "kite" form in which all the nonzero elements appear as a series of smaller symmetrical matrices along the diagonal. Operations with **P** can thus be done as smaller tasks performed on the individual smaller matrices. This means that $\mathbf{P}^{-1}$ and **Q** have

the same "kite" form as **P**. As an example, figure 2.10 shows the format of $\mathbf{P}^{-1}$ and $\{p_k(\bar{z})\}$ for the case s=3 and v=3.

$\{p_k(\bar{z})\}$	$\mathbf{P}^{-1}$																			
1	+	−	−	−	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_1z_1	−	+	−	−	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_2z_2	−	−	+	−	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_3z_3	−	−	−	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_1	0	0	0	0	+	−	−	−	0	0	0	0	0	0	0	0	0	0	0	0
$z_1z_1z_1$	0	0	0	0	−	+	−	−	0	0	0	0	0	0	0	0	0	0	0	0
$z_1z_2z_2$	0	0	0	0	−	−	+	−	0	0	0	0	0	0	0	0	0	0	0	0
$z_1z_3z_3$	0	0	0	0	−	−	−	+	0	0	0	0	0	0	0	0	0	0	0	0
z_2	0	0	0	0	0	0	0	0	+	−	−	−	0	0	0	0	0	0	0	0
$z_2z_1z_1$	0	0	0	0	0	0	0	0	−	+	−	−	0	0	0	0	0	0	0	0
$z_2z_2z_2$	0	0	0	0	0	0	0	0	−	−	+	−	0	0	0	0	0	0	0	0
$z_2z_3z_3$	0	0	0	0	0	0	0	0	−	−	−	+	0	0	0	0	0	0	0	0
z_3	0	0	0	0	0	0	0	0	0	0	0	0	+	−	−	−	0	0	0	0
$z_3z_1z_1$	0	0	0	0	0	0	0	0	0	0	0	0	−	+	−	−	0	0	0	0
$z_3z_2z_2$	0	0	0	0	0	0	0	0	0	0	0	0	−	−	+	−	0	0	0	0
$z_3z_3z_3$	0	0	0	0	0	0	0	0	0	0	0	0	−	−	−	+	0	0	0	0
z_1z_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0	0
z_1z_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0
z_2z_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0
$z_1z_2z_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+

Figure 2.10 Pattern of zero and non-zero elements of matrix $\mathbf{P}^{-1}$ for the case s=3 and v=3 (+ = positive, − = negative, 0 = zero)

An empirical method of describing the form of $\mathbf{P}^{-1}$ has been found. Consider a system of order s, having v input variables. The number of terms in $\{p_k(\bar{z})\}$ and hence the size of **P** is,

$$m(v,s) = \frac{(v+s)!}{v!s!} \qquad (2.29)$$

It is useful to define the value of this function when the arguments are zero or negative;

$$m(-v,s) \triangleq m(v,-s) \triangleq 0 \qquad (2.30)$$

$$m(0,s) \triangleq m(s,0) \triangleq 1 \qquad (2.31)$$

for v,s positive. The size and multiplicity of the smaller matrices forming $\mathbf{P}^{-1}$ can then be obtained from the terms of the identity,

$$m(v,s) = \sum_{i=1}^{s\backslash 2+1} r_i d_i \qquad (2.32)$$

where \ represents integer division and where,

$$r_i = m(v-s-1+2i, s+2-2i) \qquad (2.33)$$

$$d_i = m(v, i-1) \qquad (2.34)$$

In the matrix $\mathbf{P}^{-1}$, the number of different sizes of submatrices along the diagonal corresponds to the number of terms in the summation of (2.32). There are s\2+1 unique submatrix sizes. The size of each is given by the factor d_i and the number of occurrences of each is given by the factor r_i. The total number of nonzero elements in $\mathbf{P}^{-1}$ is,

$$nz(v,s) = \sum_{i=1}^{s\backslash 2+1} r_i d_i^{\,2} \qquad (2.35)$$

For a system of order s, nz(v,s) is much smaller than $m(v,s)^2$ for v>>s, growing in size as v is increased in manner more like that of m(v,s). This is shown for the case s=3 in figure 2.11. The matrix $\mathbf{P}^{-1}$ thus becomes almost diagonal in form for v>>s. Normalization of the input variables as in (2.23) thus significantly reduces the number of multiplications required when implementing Interference Minimization.

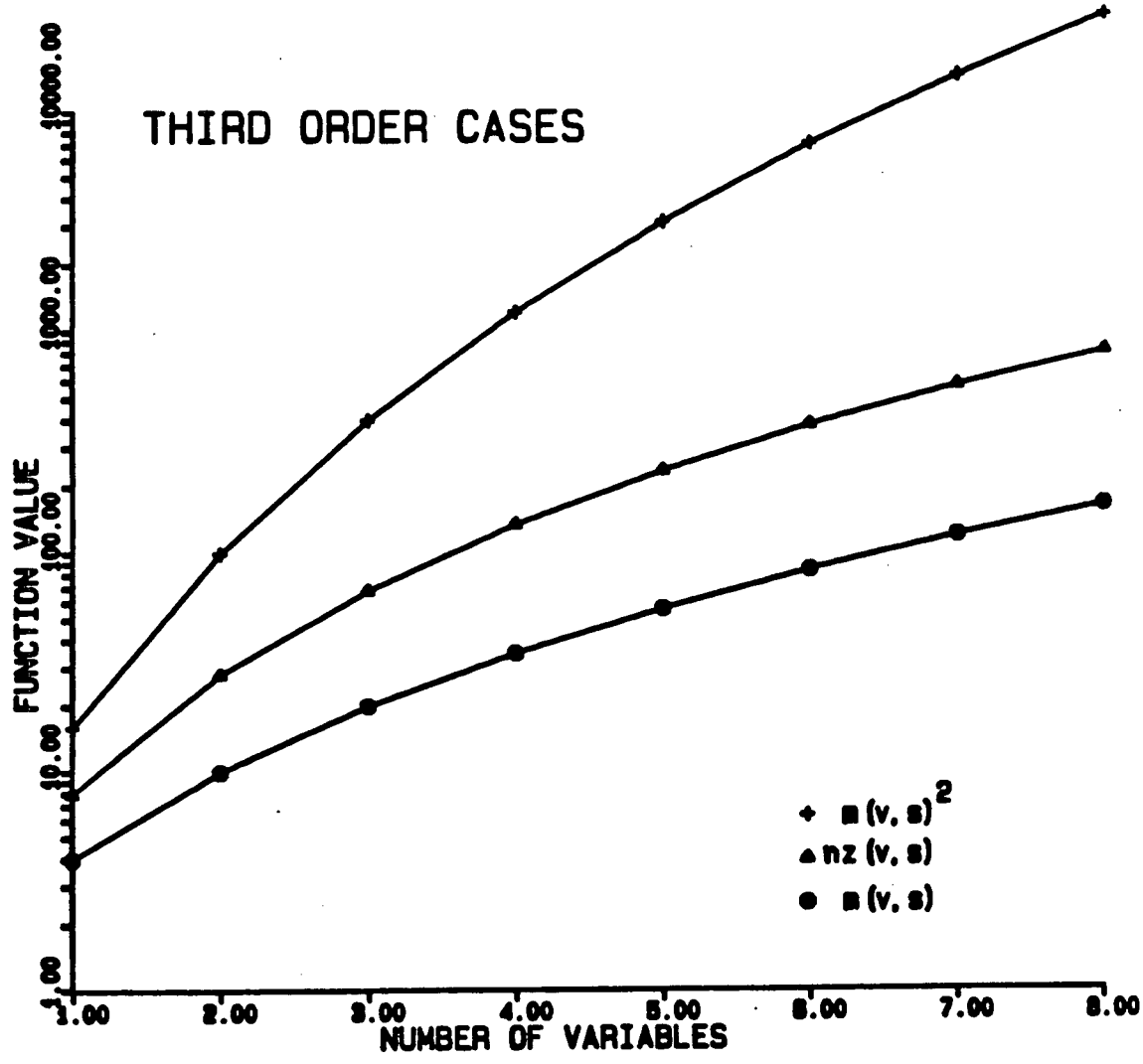


Figure 2.11 Total number of elements, number of non-zero elements and order of matrix P^{-1} for cases where s=3

2.7 APPLICATIONS WHERE POINTWISE INTERFERENCE MINIMIZATION IS APPROPRIATE

In the derivation of Interference Minimization, the error at a training point was eliminated while minimizing the change in the estimate at other points in the space. An equal weighting was assumed for these other points. A more general approach is to allow a non-uniform weighting [26]. This would be appropriate when accuracy is more critical at certain regions of the space than others. With this more general approach equation (2.7) becomes,

$$c = \int_S h(S)(\Delta\vec{w}^T\vec{p})^2 \, \delta S + a(\Delta\vec{w}^T\vec{p} - \Delta f) \qquad (2.36)$$

where h(S) is a strictly positive weighting function. The optimum weight adjustment formula remains as before except that the definition of the matrix **P** becomes,

$$\mathbf{P} = \int_S h(S)\vec{p}\vec{p}^T \, \delta S \qquad (2.37)$$

One application that demands non-uniform weighting is when the input variables have a non-uniform probability distribution and one wishes to improve the estimate most rapidly in the regions of the space that are most probable. It is then appropriate to use the probability distribution of the input variables as h(S). If h(S) is an even function of the input variables forming S then **P** is sparse as described previously.

If the distribution h(S) is not known beforehand then it is appropriate to use Pointwise Interference Minimization as it effectively learns h(S). In the limit, the matrix $\mathbf{D}_n$ converges to be a scaler multiple of the corresponding matrix **P**. For

example, if the input variables are normalized as in (2.23) then,

$$D_{\infty} = P/2 \qquad (2.38)$$

Pointwise Interference Minimization combines the learning of **P** with the learning of the estimate.

2.8 SUMMARY

A family of learning algorithms has been introduced and their convergence characteristics compared. The method of Interference Minimization has been found to be superior to either the Gradient Method or Learning Identification. Further, normalization of the input variables simplifies the implementation of Interference Minimization.

Pointwise Interference Minimization has been found to be the best algorithm on the basis of rapid convergence, however, the improvement over Interference Minimization is small and would not seem to justify the significant increase in algorithm complexity. One situation where it may be appropriate to use Pointwise Interference Minimization is when one wishes to have most rapid convergence in those regions of the input variable space that are most probable and the probability distribution, h(S), is unknown. Pointwise Interference Minimization effectively learns this distribution.

3 SELF-LEARNED CONTROL OF A TWO LINK MANIPULATOR

One of the principle contentions of this thesis is that learning algorithms can be used to achieve self-learned control of a manipulator. It had been hypothesized that weighted sums of polynomial terms could model the multi-variate, non-linear functions that describe the dynamics and kinematics of a manipulator, and that Interference Minimization or related methods could be used to learn the weighting coefficients, without recourse to analysis of the manipulator, and thus learn the functions necessary to achieve control.

3.1 RESOLVED ACCELERATION CONTROL WITH CARTESIAN PATH SPECIFICATION

Given that one has the ability to learn the multi-variate, non-linear functions describing a manipulator, it is then necessary to utilize this capability with a suitable control technique. A family of suitable methods [20,29,30] are those called variously, Inverse Plant, Inverse System, Inverse Problem , etc. One example of these methods that has been applied to manipulator control is Resolved Acceleration Control as described by Luh et al [30]. It is within this control framework that we propose to apply learning algorithms to learn manipulator control.

In general, the dynamics of a manipulator can be described by a group of functional relationships that we call the direct dynamics;

$$\ddot{\bar{a}} = \text{dir.dyn}(\bar{\tau}, \dot{\bar{a}}, \bar{a}) \tag{3.1}$$

where,

$$\dot{\bar{a}} = \int \ddot{\bar{a}}\ \delta t \tag{3.2}$$

$$\bar{a} = \int \dot{\bar{a}}\ \delta t \tag{3.3}$$

$\bar{a}$ is a vector of generalized joint angles and $\bar{\tau}$ is a vector of generalized torques or forces, as appropriate, that are applied to the joints. Application of torque $\bar{\tau}$ when the manipulator is in position $\bar{a}$ and moving with velocity $\dot{\bar{a}}$ will result in acceleration $\ddot{\bar{a}}$. The exact form of the functions comprising the direct dynamics relationships are dependent on the manipulator configuration as to type and number of links, and the link lengths, link masses and mass distributions, joint friction and gravity.

The manipulator can be controlled using its inverse dynamics which are described by a group of functional relationships that we call the inverse dynamics;

$$\bar{\tau} = \text{inv.dyn}(\ddot{\bar{a}}, \dot{\bar{a}}, \bar{a}) \tag{3.4}$$

where,

$$\ddot{\bar{a}} = \frac{\delta \dot{\bar{a}}}{\delta t}(t) \tag{3.5}$$

$$\dot{\bar{a}} = \frac{\delta \bar{a}}{\delta t}(t) \tag{3.6}$$

To achieve acceleration $\ddot{\bar{a}}$ when moving at velocity $\dot{\bar{a}}$ and in position $\bar{a}$, torque $\bar{\tau}$ must be applied. $\ddot{\bar{a}}$ and $\dot{\bar{a}}$ can be obtained be differentiating the path specification, $\bar{a}(t)$. In general it is not easy to obtain these relationships, especially for complex manipulators having several degrees of freedom.

If these relationships are known exactly, it would appear

to be possible to control a manipulator by open loop control. Given that one starts with the initially desired position equal to the actual position, then given the desired path $\bar{a}_d(t)$ one can calculate torque $\bar{\tau}(t)$ using the inverse dynamics relationships, apply $\bar{\tau}(t)$ to the manipulator and cause the manipulator to move such that $\bar{a}(t)$ equals $\bar{a}_d(t)$. This is expressed mathematically as a definition of the direct and inverse dynamics, namely,

$$\ddot{\bar{a}}(t) = \text{dir.dyn}(\text{inv.dyn}(\ddot{\bar{a}}_d,\dot{\bar{a}}_d,\bar{a}_d),\dot{\bar{a}},\bar{a}) = \ddot{\bar{a}}_d(t) \qquad (3.7)$$

given that,

$$\dot{\bar{a}}(t_0) = \dot{\bar{a}}_d(t_0) \qquad (3.8)$$

and,

$$\bar{a}(t_0) = \bar{a}_d(t_0) \qquad (3.9)$$

In practice such a control technique is poor due to finite precision in calculations, imprecise knowledge of parameters in the inverse dynamics relationships such as joint masses, joint lengths, etc. and noise-like disturbances of the manipulator by the environment. Even with a very accurate approximation of the inverse dynamics relationships and minimal environmental disturbances, tracking errors will tend to accumulate and cause further tracking errors when using open loop control.

Good control can be achieved by using feedback to adjust applied torques to correct for position and velocity errors;

$$\bar{\tau} = \text{inv.dyn}(\ddot{\bar{a}}_d+k_1(\dot{\bar{a}}_d-\dot{\bar{a}})+k_2(\bar{a}_d-\bar{a}),\dot{\bar{a}},\bar{a}) \qquad (3.10)$$

Application of such torques results in manipulator motion as follows,

$$\ddot{\bar{a}} = \text{dir.dyn}(\text{inv.dyn}(\ddot{\bar{a}}_d+k_1(\dot{\bar{a}}_d-\dot{\bar{a}})+k_2(\bar{a}_d-\bar{a}),\dot{\bar{a}},\bar{a}),\dot{\bar{a}},\bar{a}) \qquad (3.11)$$

which by the definition of the direct and inverse dynamics yields,

$$\ddot{\bar{a}} = \ddot{\bar{a}}_d + k_1(\dot{\bar{a}}_d - \dot{\bar{a}}) + k_2(\bar{a}_d - \bar{a}) \qquad (3.12)$$

$$0 = (\ddot{\bar{a}}_d - \ddot{\bar{a}}) + k_1(\dot{\bar{a}}_d - \dot{\bar{a}}) + k_2(\bar{a}_d - \bar{a}) \qquad (3.13)$$

Defining the position error as,

$$\bar{e}_a = \bar{a}_d - \bar{a} \qquad (3.14)$$

results in a second order, linear, homogeneous differential equation with constant coefficients describing the change in position error with time,

$$0 = \ddot{\bar{e}}_a + k_1\dot{\bar{e}}_a + k_2\bar{e}_a \qquad (3.15)$$

For stability, or equivalently to have $\bar{e}_a$ decrease with time, one must choose $k_1>0$, and for a critically damped response one must choose $k_1^2=4k_2$.

This technique for manipulator control has already been demonstrated by Luh et al [30] for a complex manipulator whose dynamics were known analytically. One of the main goals of this work is to determine whether this control technique can be successfully applied when the inverse dynamics relationships are learned without recourse to analysis of the dynamics of a manipulator.

In resolved acceleration control as put forward by Luh et al, the path specification is in joint coordinates. Typically, it would be more desirable to specify the manipulator path in another coordinate system which is more suitable for the task at hand. For example, one may desire to specify the path of the manipulator end point in Cartesian coordinates, rather than specify the path of the manipulator joints in joint coordinates.

Manipulator end point control in Cartesian coordinates can be achieved using the inverse kinematics of the manipulator in conjunction with the inverse dynamics of the manipulator. The inverse kinematics are described by a group of functional relationships as follows:

$$\vec{a} = \text{pos.inv.kin}(\vec{x}) \tag{3.16}$$

$$\dot{\vec{a}} = \text{vel.inv.kin}(\vec{x},\vec{a}) \tag{3.17}$$

$$\ddot{\vec{a}} = \text{acc.inv.kin}(\ddot{\vec{x}},\dot{\vec{a}},\vec{a}) \tag{3.18}$$

Given the vectors $\vec{x}$, $\dot{\vec{x}}$ and $\ddot{\vec{x}}$ describing the manipulator end point position, velocity and acceleration in Cartesian coordinates, one can obtain the vectors $\vec{a}$, $\dot{\vec{a}}$ and $\ddot{\vec{a}}$ describing the manipulator joint positions, velocities and accelerations.

Typically, the combination of joint angles that will place the manipulator end point at a given Cartesian position is not unique. It is thus necessary to apply constraints to permit the inverse kinematics to be described by functional relationships. For example, a revolute joint may be constrained to revolve through less than π radians. Such constraints often physically exist because of the manner in which manipulators are constructed; revolute joints often are restricted to less than a full circle of rotation.

For manipulators having more than three degrees of freedom, one typically wants to control more than just the end point of the manipulator. For example, with a six degree of freedom manipulator, one can specify the end point path and the orientation of the last link which is typically thought of as the hand or tool. In this case, the constraints describing the

orientation of the hand, which are necessary to permit the inverse kinematics to be described by functional relationships, can be viewed as part of the path specification rather than as constraints.

The inverse kinematics and the inverse dynamics can be combined to form a group of functional relationships that we call the Cartesian inverse dynamics;

$$\bar{\tau} = \text{cart.inv.dyn}(\ddot{\bar{x}}, \dot{\bar{x}}, \bar{x}) \tag{3.19}$$

where,

$$\ddot{\bar{x}} = \frac{\delta \dot{\bar{x}}}{\delta t}(t) \tag{3.20}$$

$$\dot{\bar{x}} = \frac{\delta \bar{x}}{\delta t}(t) \tag{3.21}$$

Similarly, the direct kinematics and the direct dynamics can be combined to form a group of functional relationships that we call the Cartesian direct dynamics;

$$\ddot{\bar{x}} = \text{cart.dir.dyn}(\bar{\tau}, \dot{\bar{x}}, \bar{x}) \tag{3.22}$$

where,

$$\dot{\bar{x}} = \int \ddot{\bar{x}}\ \delta t \tag{3.23}$$

$$\bar{x} = \int \dot{\bar{x}}\ \delta t \tag{3.24}$$

The direct kinematics of the manipulator are described by a group of functional relationships as follows,

$$\bar{x} = \text{pos.dir.kin}(\bar{a}) \tag{3.25}$$

$$\dot{\bar{x}} = \text{vel.dir.kin}(\dot{\bar{a}}, \bar{a}) \tag{3.26}$$

$$\ddot{\bar{x}} = \text{acc.dir.kin}(\ddot{\bar{a}}, \dot{\bar{a}}, \bar{a}) \tag{3.27}$$

Given the vectors $\bar{a}$, $\dot{\bar{a}}$ and $\ddot{\bar{a}}$ describing the manipulator joint positions, velocities and accelerations, one can obtain the vectors $\bar{x}$, $\dot{\bar{x}}$ and $\ddot{\bar{x}}$ describing the manipulator end point

position, velocity and acceleration.

In a manner analagous to that described previously, one can control the manipulator using the Cartesian inverse dynamics along with feedback to correct position and velocity errors;

$$\bar{\tau} = \text{cart.inv.dyn}(\ddot{\bar{x}}_d + k_1(\dot{\bar{x}}_d - \dot{\bar{x}}) + k_2(\bar{x}_d - \bar{x}), \dot{\bar{x}}, \bar{x}) \tag{3.28}$$

$$\ddot{\bar{x}} = \text{cart.dir.dyn}(\text{cart.inv.dyn}(\ddot{\bar{x}}_d + k_1(\dot{\bar{x}}_d - \dot{\bar{x}}) + k_2(\bar{x}_d - \bar{x}), \dot{\bar{x}}, \bar{x}), \dot{\bar{x}}, \bar{x}) \tag{3.29}$$

$$\ddot{\bar{x}} = \ddot{\bar{x}}_d + k_1(\dot{\bar{x}}_d - \dot{\bar{x}}) + k_2(\bar{x}_d - \bar{x}) \tag{3.30}$$

$$0 = (\ddot{\bar{x}}_d - \ddot{\bar{x}}) + k_1(\dot{\bar{x}}_d - \dot{\bar{x}}) + k_2(\bar{x}_d - \bar{x}) \tag{3.31}$$

Defining the Cartesian position error as,

$$\bar{e}_x = \bar{x}_d - \bar{x} \tag{3.32}$$

one can choose k_1 and k_2 as before to achieve stability and critical damping of Cartesian position errors.

The Cartesian inverse dynamics relationships are more complex than the inverse arm relationships due to the inclusion of a coordinate transformation through use of the inverse kinematics. Such would be the case if one wanted to use this approach in conjunction with other coordinate systems, such as cylindrical or spherical, that might be more convenient given the task at hand. The key point is that control is desired with respect to a coordinate system that may not be the most natural one for describing manipulator dynamics. A second major goal of the work is thus to determine whether learned resolved acceleration control can be successfully applied using path specification in a task oriented coordinate system such as Cartesian coordinates.

When applying resolved acceleration control using

Cartesian coordinates for end point path specification, it is necessary to observe the end point Cartesian position in order to use feedback to correct position and velocity errors. Ideally, one could use a vision system to measure the end point path. Equivalently, one can measure joint positions, velocities and accelerations and use the direct kinematic relationships to determine the manipulator end point path. The direct kinematics can be determined by a general technique [44,46] and they are much simpler to derive than the inverse kinematics. Their evaluation requires nothing more than multiplication and summation of sine and cosine terms. Nevertheless, such a method requires manipulator analysis and time consuming, manipulator-specific calculations.

With Interference Minimization it should be possible to learn the functional relationships that describe the direct kinematics. This would allow Cartesian path control of manipulator without the use of a vision system and without resort to analysis and calculation of the direct kinematics. The manipulator end point could then be obtained from simple joint path measurements.

3.2 A TWO LINK MANIPULATOR

In order to test the feasibility of learned control of a manipulator, a simple two link manipulator was chosen as a test vehicle. The structure of the two link manipulator and associated coordinate reference frames are shown in figure 3.1.

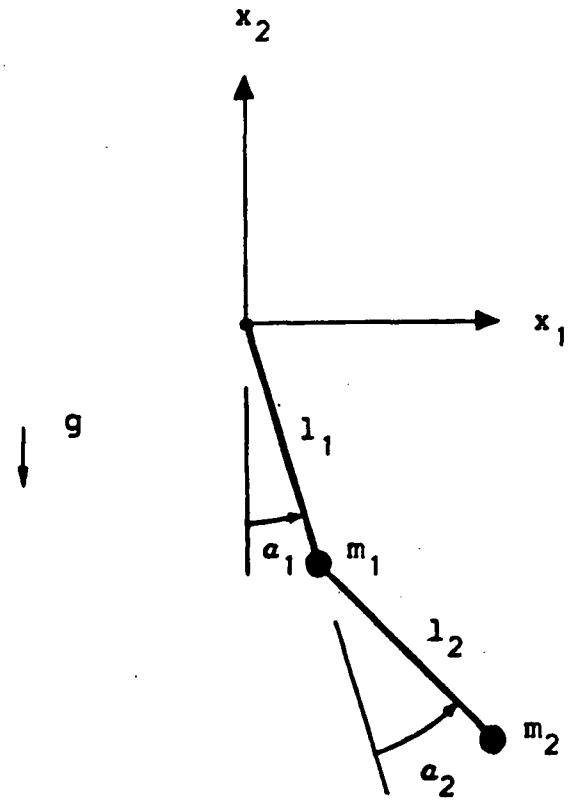


Figure 3.1 A two link manipulator

Such a simple manipulator was chosen for two basic reasons: First, its dynamics and kinematics are known in analytical form. This permits the arm to be simulated on a computer, with all the advantages that simulation brings; no time consuming hardware development, exact specification of parameters, accurate observation of all variables, and no physical damage when mistakes are made. Also, control performance when using the analytical dynamics and kinematics can serve as benchmarks against which one can compare control performances when using learned dynamics and kinematics. Secondly, as the manipulator was to be simulated, it had to be simple to avoid requiring inordinate amounts of computer time for accurate simulation of

numerous movements. In spite of the simplicity of the two link manipulator, its kinematics and dynamics are described by highly non-linear, multi-variate functions as is typical of more complex manipulators. We expect that results using the two link manipulator will be representative of possible results using more complex manipulators.

The direct kinematics of the two link manipulator can be obtained by using basic trigonometry to calculate the Cartesian position of the end point as a function of the joint angles. These functions are then differentiated to yield the Cartesian velocity and acceleration of the end point in terms of joint angles, angular velocities and angular accelerations. The direct kinematics are as follows:

$$x_1 = l_1\sin(a_1) + l_2\sin(a_1+a_2) \tag{3.33}$$

$$x_2 = -l_1\cos(a_1) - l_2\cos(a_1+a_2) \tag{3.34}$$

$$\dot{x}_1 = l_1\cos(a_1)\dot{a}_1 + l_2\cos(a_1+a_2)(\dot{a}_1+\dot{a}_1) \tag{3.35}$$

$$\dot{x}_2 = l_1\sin(a_1)\dot{a}_1 + l_2\sin(a_1+a_2)(\dot{a}_1+\dot{a}_2) \tag{3.36}$$

$$\ddot{x}_1 = l_1\cos(a_1)\ddot{a}_1 + l_2\cos(a_1+a_2)(\ddot{a}_1+\ddot{a}_2) - l_1\sin(a_1)\dot{a}_1^2 - l_2\sin(a_1+a_2)(\dot{a}_1+\dot{a}_2)^2 \tag{3.37}$$

$$\ddot{x}_2 = l_1\sin(a_1)\ddot{a}_1 + l_2\sin(a_1+a_2)(\ddot{a}_1+\ddot{a}_2) + l_1\cos(a_1)\dot{a}_1^2 + l_2\cos(a_1+a_2)(\dot{a}_1+\dot{a}_2)^2 \tag{3.38}$$

To obtain the inverse kinematics of the two link manipulator it is necessary to apply a constraint to make functional relationships possible. In the following we restrict a_2 to be between 0 and π radians. The inverse kinematics can then be written as,

$$a_2 = \arccos((x_1^2 + x_2^2 - l_1^2 - l_2^2)/2l_1l_2) \tag{3.39}$$

$$a_1 = \arctan(x_1,-x_2) - \arctan(l_2\sin(a_2),l_1 + l_2\cos(a_2)) \tag{3.40}$$

$$\dot{a}_1 = (l_2\sin(a_1+a_2)\dot{x}_1 - l_2\cos(a_1+a_2)\dot{x}_2)/l_1l_2\sin(a_2) \tag{3.41}$$

$$\dot{a}_2 = [(-l_1\sin(a_1)\dot{x}_1 + l_1\cos(a_1)\dot{x}_2)/l_1l_2\sin(a_2)] - \dot{a}_1 \tag{3.42}$$

$$\ddot{a}_1 = (l_2\sin(a_1+a_2)\ddot{x}_1 - l_2\cos(a_1+a_2)\ddot{x}_2 + l_1l_2\cos(a_2)\dot{a}_1^2 + l_2^2(\dot{a}_1+\dot{a}_2)^2)/l_1l_2\sin(a_2) \tag{3.43}$$

$$\ddot{a}_2 = [(-l_1\sin(a_1)\ddot{x}_1 + l_1\cos(a_1)\ddot{x}_2 - l_1^2\dot{a}_1^2 - l_1l_2\cos(a_2)(\dot{a}_1+\dot{a}_2)^2)/l_1l_2\sin(a_2)] - \ddot{a}_1 \tag{3.44}$$

Note that arctan function has two arguments and is thus assumed to yield a value between $-\pi$ and π. Since it is the sines and cosines of a_1 and a_2 that are used in subsequent calculations, it is useful to use the following formulas,

$$\cos(a_2) = (x_1^2 + x_2^2 - l_1^2 - l_2^2)/2l_1l_2 \tag{3.45}$$

$$\sin(a_2) = (1 - \cos(a_2)^2)^{1/2} \tag{3.46}$$

$$\cos(a_1) = [x_1l_1\sin(a_2) - x_2(l_1+l_2\cos(a_2))]/(x_1^2+x_2^2) \tag{3.47}$$

$$\sin(a_1) = (1 - \cos(a_1)^2)^{1/2} \tag{3.48}$$

The derivation of the direct and inverse kinematics is shown in appendix A.

The inverse dynamics of the two link manipulator can be obtained using Lagrangian mechanics [37,46]. The Lagrangian L is defined as the difference between the kinetic energy K and the potential energy P of the system,

$$L = K - P \tag{3.49}$$

The inverse dynamics equations are obtained as,

$$\tau_i = \frac{\delta}{\delta t}\left[\frac{\partial L}{\partial \dot{a}_i}\right] - \frac{\partial L}{\partial a_i} \tag{3.50}$$

where a_i are the coordinates in which the kinetic and potential energy are expressed, $\dot{a}_i$ are the corresponding velocities, and τ_i the corresponding forces or torques. τ_i is either a force or a torque, depending upon whether a_i is a linear or angular coordinate. These forces, torques and coordinates are referred to as generalized forces, torques and coordinates.

Application of Lagrangian mechanics to a two link manipulator having link lengths l_1 and l_2, link masses m_1 and m_2, and oriented as shown in figure 3.1 with respect to g, the gravitational field, yields the following inverse dynamics,

$$\tau_1 = d_{11}\ddot{a}_1 + d_{12}\ddot{a}_2 + d_{111}\dot{a}_1^2 + d_{122}\dot{a}_2^2 + d_{112}\dot{a}_1\dot{a}_2 + d_{121}\dot{a}_2\dot{a}_1 + d_1 \tag{3.51}$$

$$\tau_2 = d_{21}\ddot{a}_1 + d_{22}\ddot{a}_2 + d_{211}\dot{a}_1^2 + d_{222}\dot{a}_2^2 + d_{212}\dot{a}_1\dot{a}_2 + d_{221}\dot{a}_2\dot{a}_1 + d_2 \tag{3.52}$$

where,

$$d_{11} = (m_1+m_2)l_1^2 + m_2l_2^2 + 2m_2l_1l_2\cos(a_2) \tag{3.53}$$

$$d_{12} = m_2l_2^2 + m_2l_1l_2\cos(a_2) \tag{3.54}$$

$$d_{111} = 0 \tag{3.55}$$

$$d_{122} = -m_2l_1l_2\sin(a_2) \tag{3.56}$$

$$d_{112} = -m_2l_1l_2\sin(a_2) \tag{3.57}$$

$$d_{121} = -m_2l_1l_2\sin(a_2) \tag{3.58}$$

$$d_1 = (m_1+m_2)gl_1\sin(a_1) + m_2gl_2\sin(a_1+a_2) \tag{3.59}$$

$$d_{21} = m_2l_2^2 + m_2l_1l_2\cos(a_2) \tag{3.60}$$

$$d_{22} = m_2l_2^2 \tag{3.61}$$

$$d_{211} = m_2l_1l_2\sin(a_2) \tag{3.62}$$

$$d_{222} = 0 \tag{3.63}$$

$$d_{212} = 0 \tag{3.64}$$

$$d_{221} = 0 \tag{3.65}$$

$$d_2 = m_2 g l_2 \sin(a_1+a_2) \tag{3.66}$$

The derivation of the inverse dynamics is shown in appendix B and follows the approach used by Paul [46]. Note, however, that the results differ slightly from those given by Paul as there is an arithmetic error in the his derivation. Note also that this derivation neglects frictional forces. This is not necessarily a simplification, though, from a control point of view. Friction would tend to damp any oscillations that occur whereas with no friction, oscillations must be damped out by the control system.

The direct dynamics can be obtained through algebraic manipulation of the inverse dynamics and are as follows,

$$\ddot{a}_1 = [d_{12}(\tau_2 - g_2) + d_{22}(g_1 - \tau_1)]/d_0 \tag{3.67}$$

$$\ddot{a}_2 = [d_{21}(\tau_1 - g_1) + d_{11}(g_2 - \tau_2)]/d_0 \tag{3.68}$$

where,

$$d_0 = d_{12}d_{21} - d_{11}d_{22} \tag{3.69}$$

$$g_1 = d_{111}\dot{a}_1^2 + d_{122}\dot{a}_2^2 + d_{112}\dot{a}_1\dot{a}_2 + d_{121}\dot{a}_2\dot{a}_1 + d_1 \tag{3.70}$$

$$g_2 = d_{211}\dot{a}_1^2 + d_{222}\dot{a}_2^2 + d_{212}\dot{a}_1\dot{a}_2 + d_{221}\dot{a}_2\dot{a}_1 + d_2 \tag{3.71}$$

It is the direct dynamics in conjunction with the integral equations,

$$\dot{\vec{a}} = \int \ddot{\vec{a}}\ \delta t \tag{3.72}$$

$$\vec{a} = \int \dot{\vec{a}}\ \delta t \tag{3.73}$$

that allows simulation of the manipulator on a computer utilizing numerical integration routines.

To achieve learned resolved acceleration control in a

Cartesian reference frame it is necessary to learn the inverse dynamics and the inverse kinematics. To replace a vision system or its equivalent for feedback purposes, it is necessary to learn the direct kinematics. It can be seen that even for a simple two link manipulator the functional relationships to be learned are multi-variate and highly non-linear.

3.3 SIMULATION OF THE MANIPULATOR

Testing of various control systems was carried out using a simulation of the two link manipulator. The simulation was performed on a VAX-11/750 computer using ACSL (advanced continuous simulation language). ACSL permits the simulation of continuous time systems, such as the manipulator, as well as simulation of discrete time systems, such as the digital computing devices that would be used to implement a manipulator control system [58]. A schematic diagram of the simulation is shown in figure 3.2.

The various blocks in figure 3.2 can be thought of as separate physical entities. Derivative blocks represent continuous time systems; systems whose states change continuously, whereas discrete blocks represent discrete time systems; systems whose states change at discrete points in time.

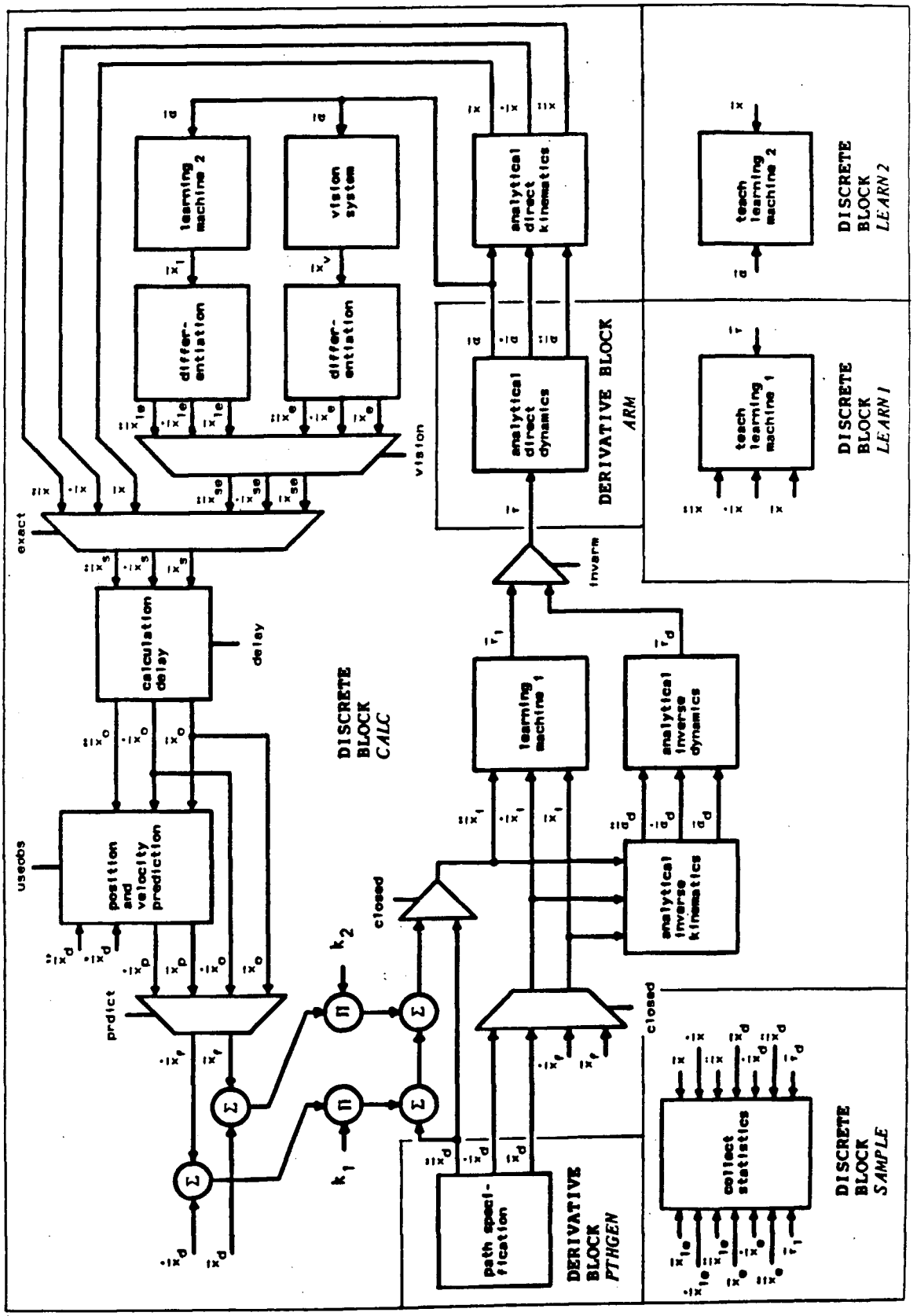


Figure 3.2 Block diagram of manipulator simulation program

DERIVATIVE BLOCK ARM

Derivative block ARM represents the two link manipulator. Given a statement of the direct dynamics in differential form,

$$\ddot{\bar{a}} = \text{dir.dyn}(\bar{\tau},\dot{\bar{a}},\bar{a}) \tag{3.74}$$

the integration algorithms of ACSL advance the states of the system, $\dot{\bar{a}}$ and $\bar{a}$, as,

$$\dot{\bar{a}} = \int \ddot{\bar{a}}\ \delta t \tag{3.75}$$

$$\bar{a} = \int \dot{\bar{a}}\ \delta t \tag{3.76}$$

Of the various integration algorithms available in ACSL, the Adams-Moulton variable stepsize, variable order algorithm was chosen. The relative error introduced at each integration step was restricted to be less than 1.0E-6 times the maximum value of the state up to that point in the simulation and the absolute error introduced at each integration step was restricted to be less than 1.0E-6. Since the order of magnitude of the states is about 1 and FORTRAN single precision real variables are used by ACSL for storage, these error bounds are near the limit of what is achievable. Other fixed stepsize, fixed order integration algorithms available in ACSL, such as Euler, 2nd order Runge-Kutta, and 4th order Runge-Kutta, typically required more CPU time to achieve comparable error bounds. The Gear's Stiff variable stepsize, variable order algorithm was not considered as it is only appropriate for certain difficult integration problems, which is not the case here [58]. ACSL ensures that states and derivatives in derivative blocks are up to date at the times when any other blocks are being executed. Thus the simulated two link manipulator appears as a continuous system to

the rest of the simulation.

DERIVATIVE BLOCK PTHGEN

Derivative block PTHGEN contains code to compute the current path specification, $\bar{x}_d$, $\dot{\bar{x}}_d$ and $\ddot{\bar{x}}_d$, given the current time, t. No integration is performed by PTHGEN as a dynamic system with states is not being modelled here. A derivative block is used in this manner simply to ensure that the path specification is current whenever another block is being executed without having to duplicate the code in each block that uses the path specification. Paths can be either linear or circular as described in section 3.4.

DERIVATIVE BLOCK CALC

Discrete block CALC performs the computations required to control the manipulator. Inputs are the specification of the desired Cartesian path, $\bar{x}_d$, $\dot{\bar{x}}_d$ and $\ddot{\bar{x}}_d$, and the view of the resulting joint path, $\bar{a}$, $\dot{\bar{a}}$ and $\ddot{\bar{a}}$. The output is the torque $\bar{\tau}$ that is applied to the manipulator. Since CALC is a discrete block, it is executed as discrete intervals in time as fixed by t_{calc}. The output torque is thus updated at intervals of t_{calc} and held constant otherwise. The time required to perform the computations of the CALC block would determine the lower bound on t_{calc} in an actual implementation of the control system with a real manipulator.

The many subparts of discrete block CALC allow various types of control to be performed. Control can be open loop, using only the desired Cartesian path specification or closed loop, using the view of the resulting joint path for error

correcting feedback. Logical variable CLOSED determines whether control is open or closed loop. Since path specification is in Cartesian coordinates, feedback information must also be in Cartesian coordinates. An approximate Cartesian view of the resulting path can be determined from the resulting joint path by using a model of vision system or by using a learned equivalent of the vision system based on Learning Machine 2. Selection is made according to logical variable VISION. Alternatively, the exact resulting Cartesian path can be obtained using the analytical direct kinematics if logical variable EXACT is true. Due to calculation delay in a real application, control using feedback cannot make use of current observations of the resulting path but rather only observations from at least time t_{calc} previous. A delay of t_{calc} in the observations of the resulting path is implemented when logical variable DELAY is true. Given that there is a delay in observations, one can attempt to predict what the current observations would be. Such prediction is done whenever logical variable PRDICT is true. Position and velocity prediction is done using either the desired velocity and acceleration or the observed velocity and acceleration depending on the state of logical variable USEOBS. Finally, one can calculate the torque to apply to the manipulator by using the analytical Cartesian inverse dynamics, consisting of the analytical inverse kinematics and the analytical inverse dynamics, or by using a learned equivalent of the Cartesian inverse dynamics, Learning Machine 1. The method of calculation is selected by logical

variable INVARM, Learning Machine 1 being selected when INVARM is false.

DISCRETE BLOCKS LEARN1 AND LEARN2

Discrete blocks LEARN1 and LEARN2 contain code that trains Learning Machines 1 and 2, respectively, using observations of the system. These blocks are executed at intervals of t_{lrn1} and t_{lrn2} when enabled by logical variables LRNINV and LRNVIS, respectively.

DISCRETE BLOCK SAMPLE

Discrete block SAMPLE contains code to collect statistical information about the performance of the control system. It is executed at intervals of t_{samp}. t_{samp} was set to be 0.01 sec for all simulations. Information compiled includes root mean square path error,

$$\text{XERMS} = (\text{mean}\{(x_i - x_{di})^2\})^{1/2} \tag{3.77}$$

$$\text{VXERMS} = (\text{mean}\{(\dot{x}_i - \dot{x}_{di})^2\})^{1/2} \tag{3.78}$$

$$\text{AXERMS} = (\text{mean}\{(\ddot{x}_i - \ddot{x}_{di})^2\})^{1/2} \tag{3.79}$$

root mean square path estimation error,

$$\text{EXERMS} = (\text{mean}\{(x_i - x_{ei})^2\})^{1/2} \tag{3.80}$$

$$\text{EVERMS} = (\text{mean}\{(\dot{x}_i - \dot{x}_{ei})^2\})^{1/2} \tag{3.81}$$

$$\text{EAERMS} = (\text{mean}\{(\ddot{x}_i - \ddot{x}_{ei})^2\})^{1/2} \tag{3.82}$$

root mean square learned path estimation error,

$$\text{LXERMS} = (\text{mean}\{(x_i - x_{lei})^2\})^{1/2} \tag{3.83}$$

$$\text{LVERMS} = (\text{mean}\{(\dot{x}_i - \dot{x}_{lei})^2\})^{1/2} \tag{3.84}$$

$$\text{LAERMS} = (\text{mean}\{(\ddot{x}_i - \ddot{x}_{lei})^2\})^{1/2} \tag{3.85}$$

root mean square torque estimation error,

$$\text{TERMS} = (\text{mean}\{(\tau_{di} - \tau_{li})^2\})^{1/2} \tag{3.86}$$

and maximum applied torque,

$$TRQMAX = \max\{abs(\tau_i)\} \qquad (3.87)$$

Also compiled are statistics about how often and how far the resulting path motion is out of bounds, as bounds on allowable motion are imposed by certain of the control techniques considered.

More detailed descriptions of certain aspects of the simulation are included in following sections when such aspects of the simulation are central to the particular control technique being studied.

3.4 PATH SPECIFICATION

Before going on to discuss methods of controlling the manipulator it is necessary to describe how path specifications are generated. Two types of paths are used to test control of the manipulator; linear paths and circular paths.

LINEAR PATHS

Linear paths simply consist of linear motion between two points given by the coordinates $<x_{in1},x_{in2}>$ and $<x_{fi1},x_{fi2}>$. The path consists of five phases of motion. No motion occurs until time t_0, then there is a phase of constant acceleration, a_{max}, a phase of constant velocity, v_{max}, a phase of constant decelaration, a_{max}, and finally a period of no motion, again of duration t_0. If the initial and final points are close together then v_{max} is not reached. This occurs when,

$$[(x_{fi1}-x_{in1})^2 + (x_{fi2}-x_{in2})^2]^{1/2} < v_{max}^2/a_{max} \qquad (3.88)$$

In this case there is no phase of constant, non-zero velocity,

rather there is simply acceleration until halfway along the path and then deceleration until stopped.

Four standard linear paths were used for test purposes. The data defining these four standard paths is given in table 3.1. A plot of line 1 in Cartesian coordinates is shown in figure 3.3. Figure 3.4 shows line 1 in joint coordinates.

PATH	x_{in1}	x_{in2}	x_{fi1}	x_{fi2}	v_{max}	a_{max}	t_0
line 1	0.45	-1.45	-0.45	-0.55	0.80	0.80	0.50
line 2	0.45	-0.55	-0.45	-1.45	0.80	0.80	0.50
line 3	-0.45	-1.00	0.45	-1.00	0.80	0.80	0.50
line 4	0.00	-1.45	0.00	-0.55	0.80	0.80	0.50

Table 3.1 Data defining standard linear paths

CIRCULAR PATHS

Circular paths are centered about the origin of the work space located at coordinates $\langle x_{c1}, x_{c2} \rangle = \langle 0, -1 \rangle$. Motion can be clockwise or counterclockwise. The initial position is given by the specified radius, ρ, and the specified starting angle, θ, as follows,

$$x_{in1} = x_{c1} + \rho\cos(\theta) \qquad (3.89)$$

$$x_{in2} = x_{c2} + \rho\sin(\theta) \qquad (3.90)$$

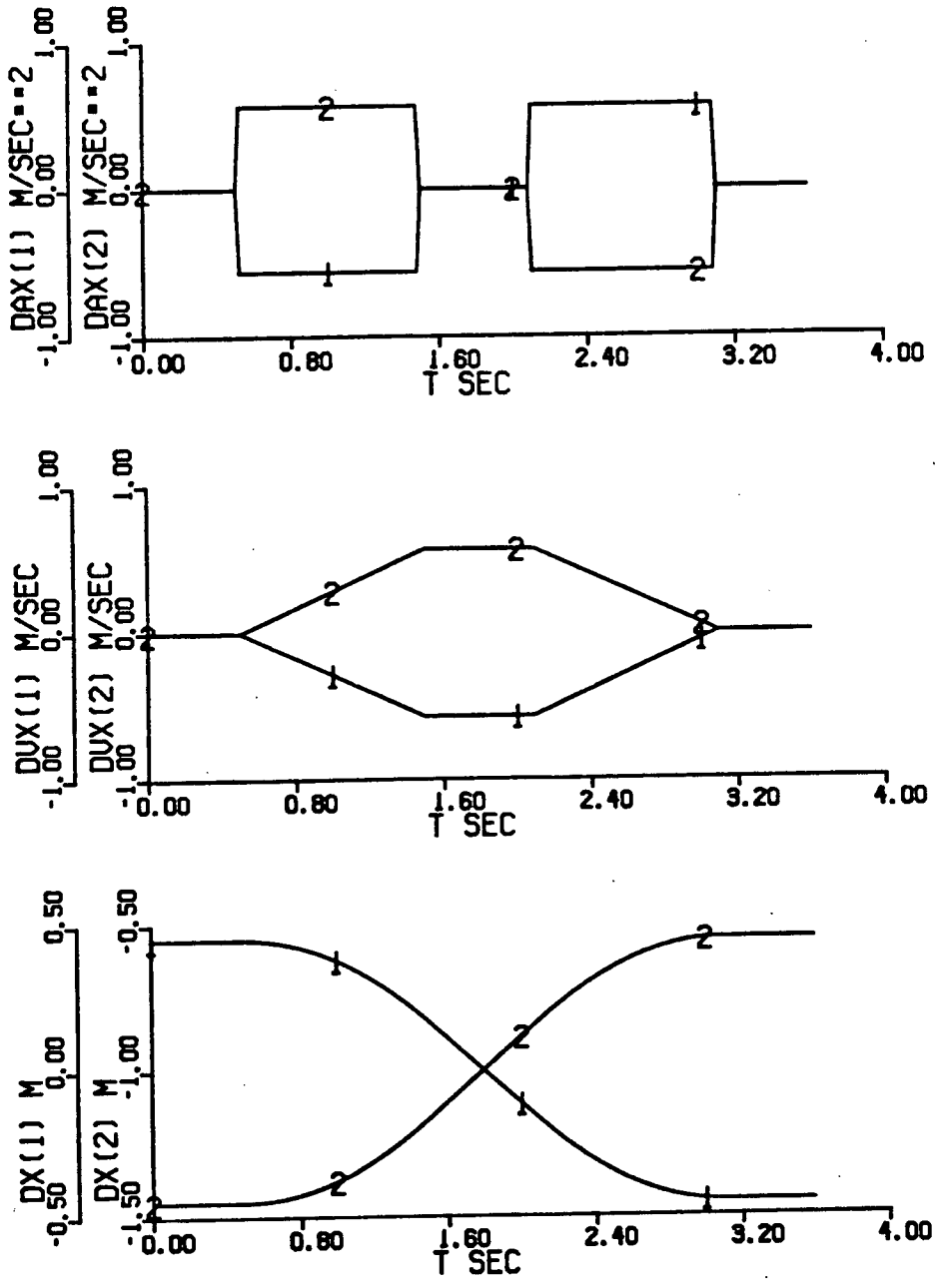


Figure 3.3 Standard path line 1 in Cartesian coordinates

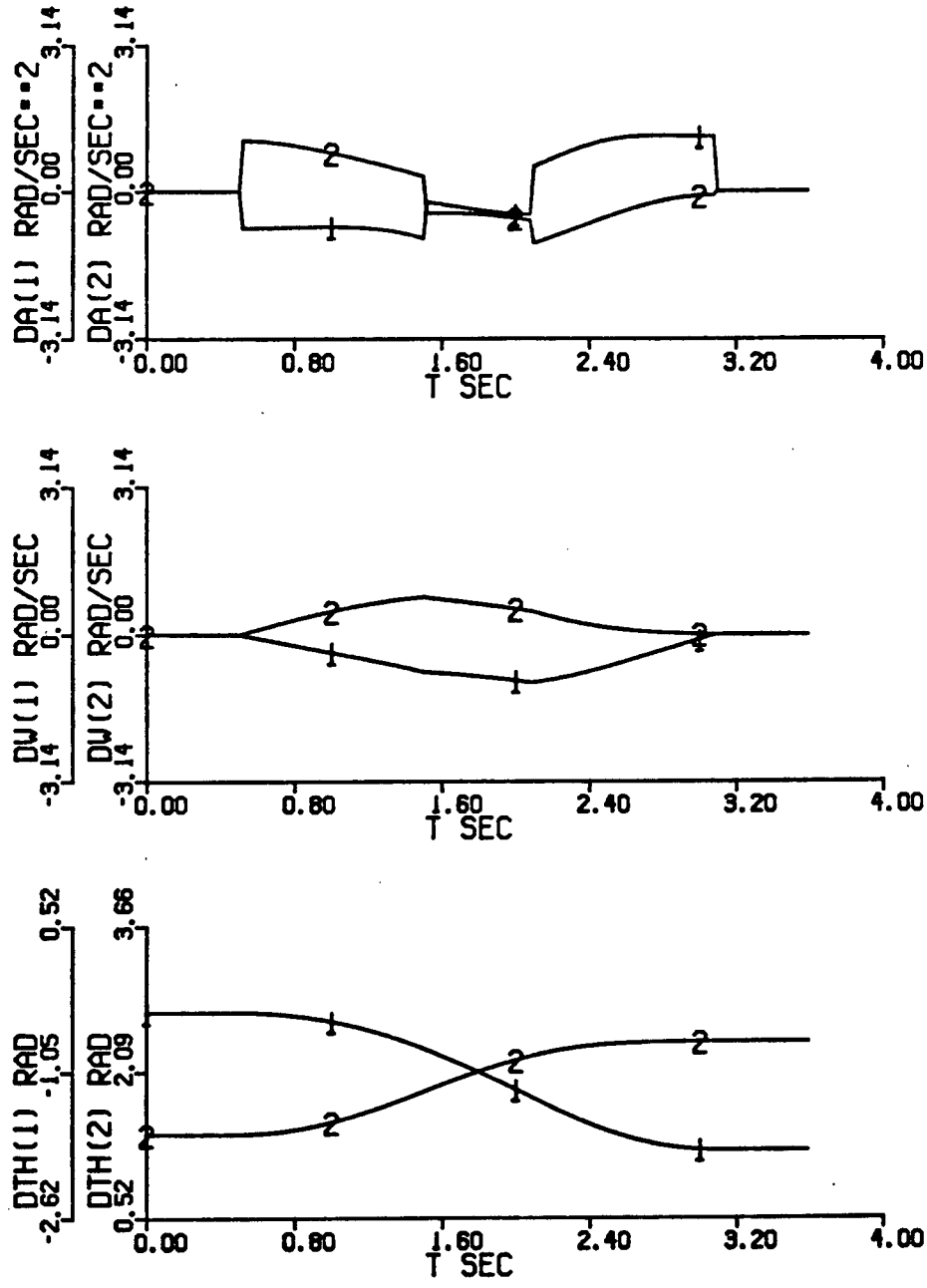


Figure 3.4 Standard path line 1 in joint coordinates

Motion proceeds along a circular arc for the fraction of a complete revolution specified by β. Along the tangent, the same five phases of motion occur as with linear paths, except that v_{tng} and a_{tng} replace v_{max} and a_{max}, respectively. The maximum tangential velocities and accelerations must be constrained, however, such that the vector sum of tangential and radial acceleration never exceeds a_{max}. This is accomplished by choosing,

$$v_{tng} = \min \{v_{max}, (\rho\gamma a_{max})^{1/2}\} \qquad (3.91)$$

$$a_{tng} = (a_{max}^{2} - v_{tng}^{4}/\rho^{2})^{1/2} \qquad (3.92)$$

where γ is a factor that specifies the portion of a_{max} that is to be allocated to centripetal acceleration during the constant tangential velocity phase of motion.

Two circular paths were used for test purposes. The data defining these two standard paths is given in table 3.2. A plot of circle 1 in Cartesian coordinates is shown in figure 3.5. Figure 3.6 shows circle 1 in joint coordinates and figure 3.7 shows a plan view of circle 1.

PATH	ρ	θ	β	γ	v_{max}	a_{max}	t_0	dir
circle 1	0.40	0.00	1.00	0.75	0.90	0.90	1.00	cw
circle 2	0.40	3.14	1.00	0.75	0.90	0.90	1.00	ccw

Table 3.2 Data defining standard circular paths

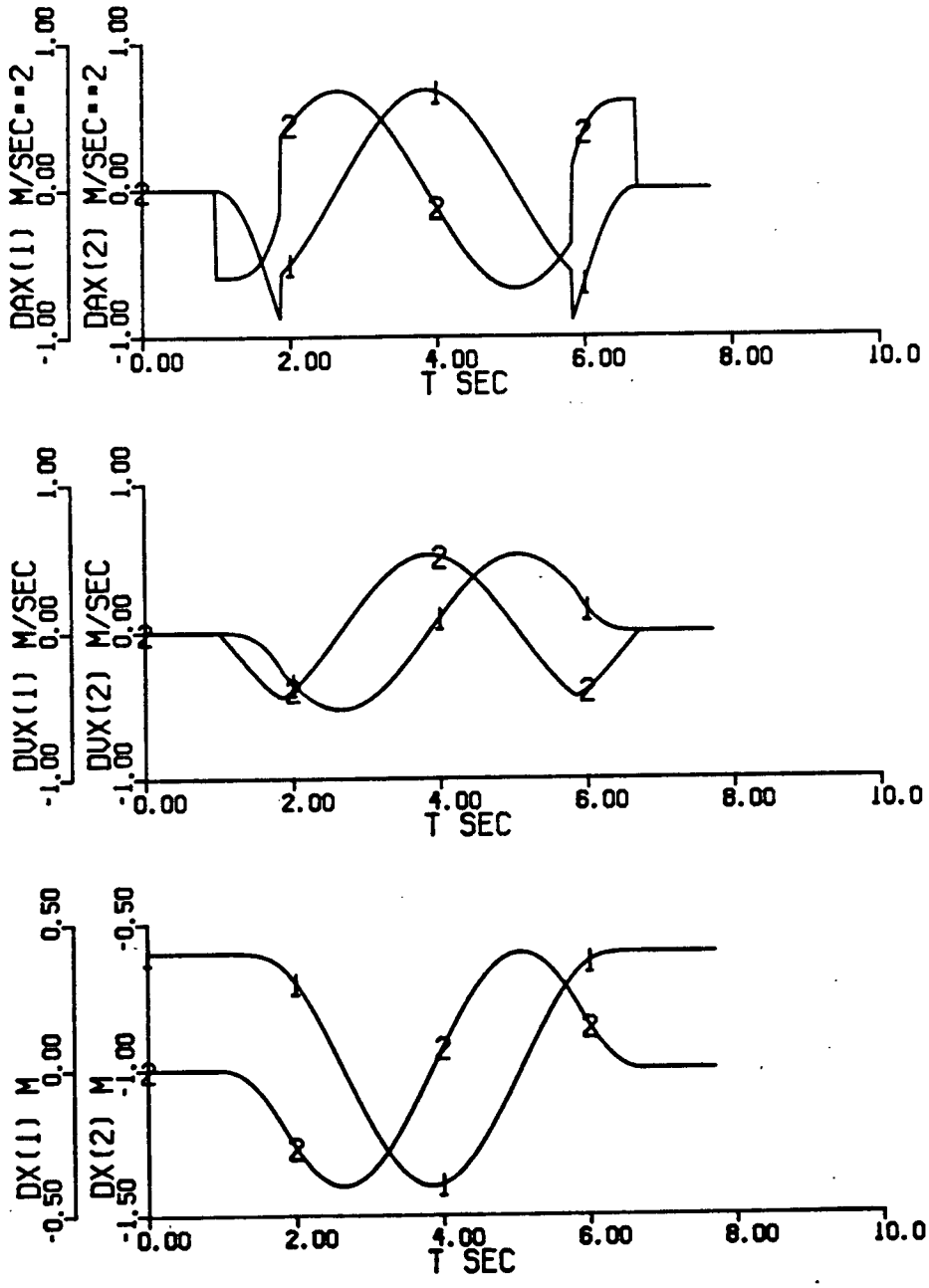


Figure 3.5 Standard path circle 1 in Cartesian coordinates

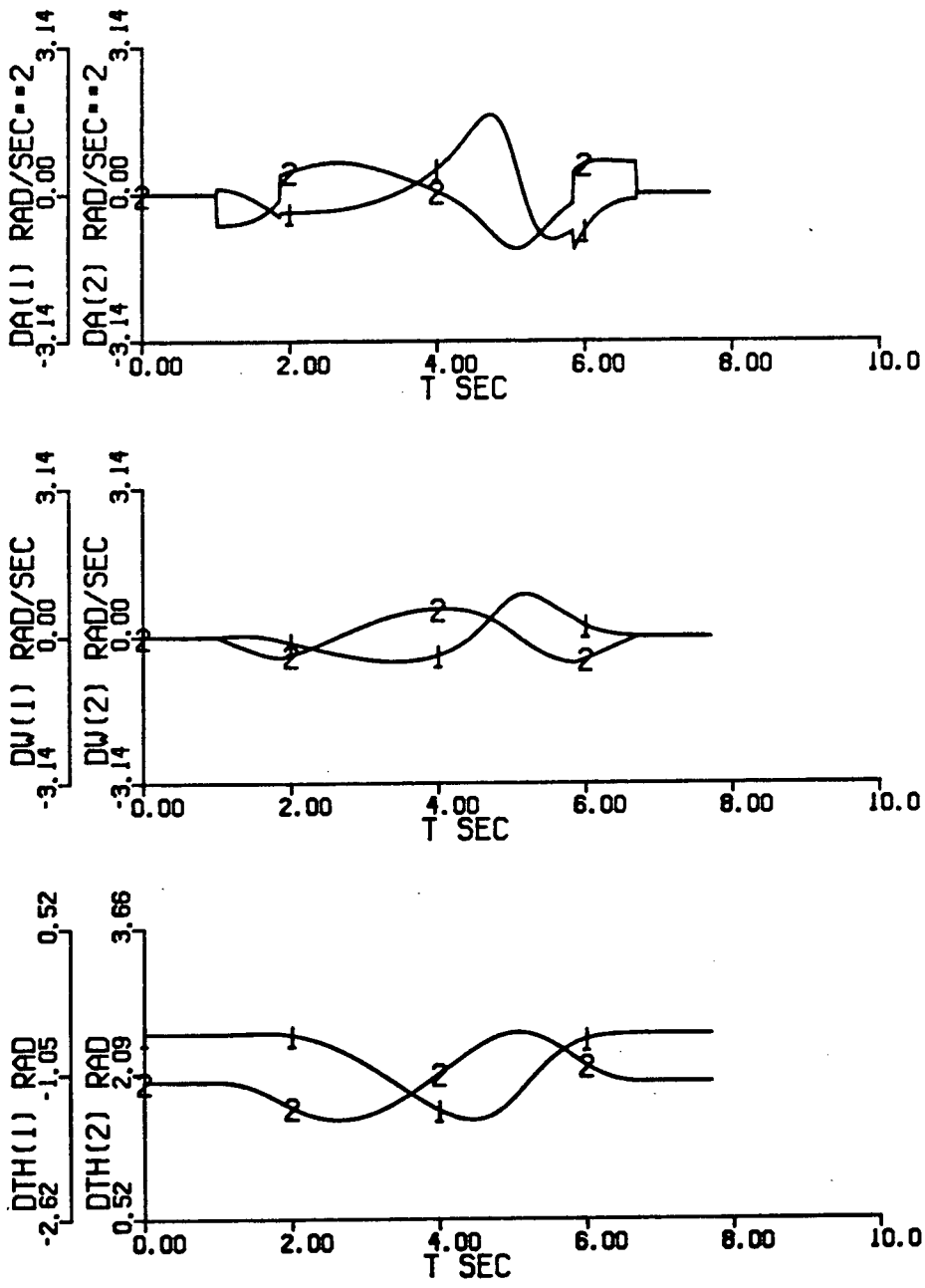


Figure 3.6 Standard path circle 1 in joint coordinates

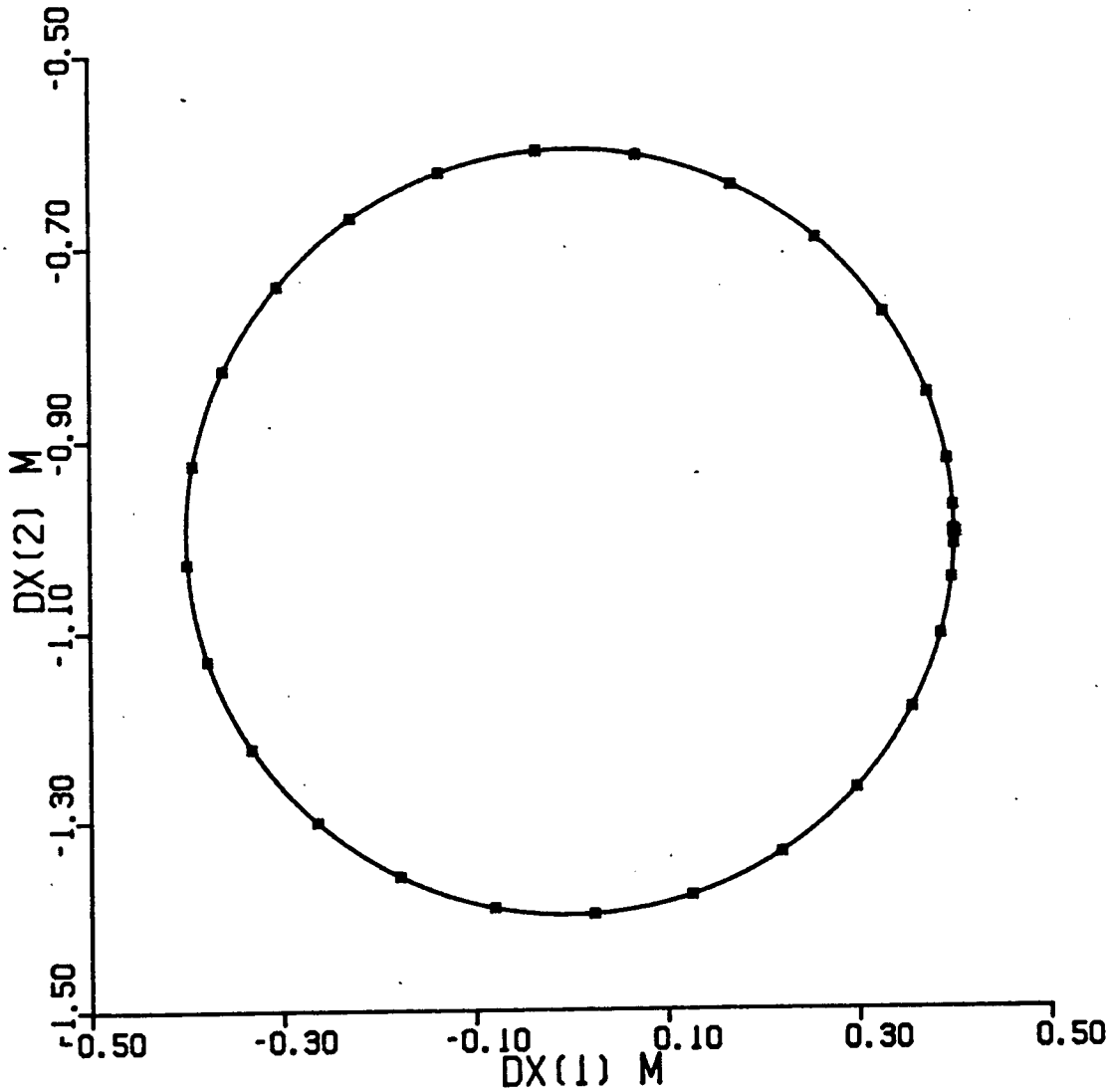


Figure 3.7 View of standard path circle 1

(ticks mark intervals of 0.2 sec)

3.5 MANIPULATOR CONTROL USING THE ANALYTICAL INVERSE DYNAMICS AND ANALYTICAL INVERSE KINEMATICS

Manipulator control using the analytical inverse dynamics and analytical inverse kinematics, which are together called the analytical Cartesian inverse dynamics, warrants investigation as it forms a benchmark against which to compare manipulator control using a learned sum of polynomials representation of the Cartesian inverse dynamics.

3.5.1 IDEAL OPEN LOOP CONTROL

Before moving on to more realistic control scenarios, it is instructive to simulate ideal open loop control of the manipulator. The meaning of open loop control is clear, however, ideal requires explanation. By ideal, we mean that the analytical inverse kinematics and analytical inverse dynamics are used to compute the torques required to achieve a given path. It also means that the torque is updated so frequently that it is effectively calculated continuously. To test ideal open loop control, two simulations were carried out with CLOSED=.FALSE., INVARM=.TRUE. and t_{calc}=0.0001 sec. Since the duration of motion for the paths was several seconds, such a small t_{calc} assures that torque is effectively calculated continuously. Table 3.3 shows the path error that occurred for the paths, line 1 and circle 1, and figure 3.8 shows the resulting path in Cartesian coordinates for the simulation of line 1. The errors that occur are minimal for both paths and the resulting path for line 1 is indistinguishable from the desired path, which is shown in figure 3.3. The errors can be attributed

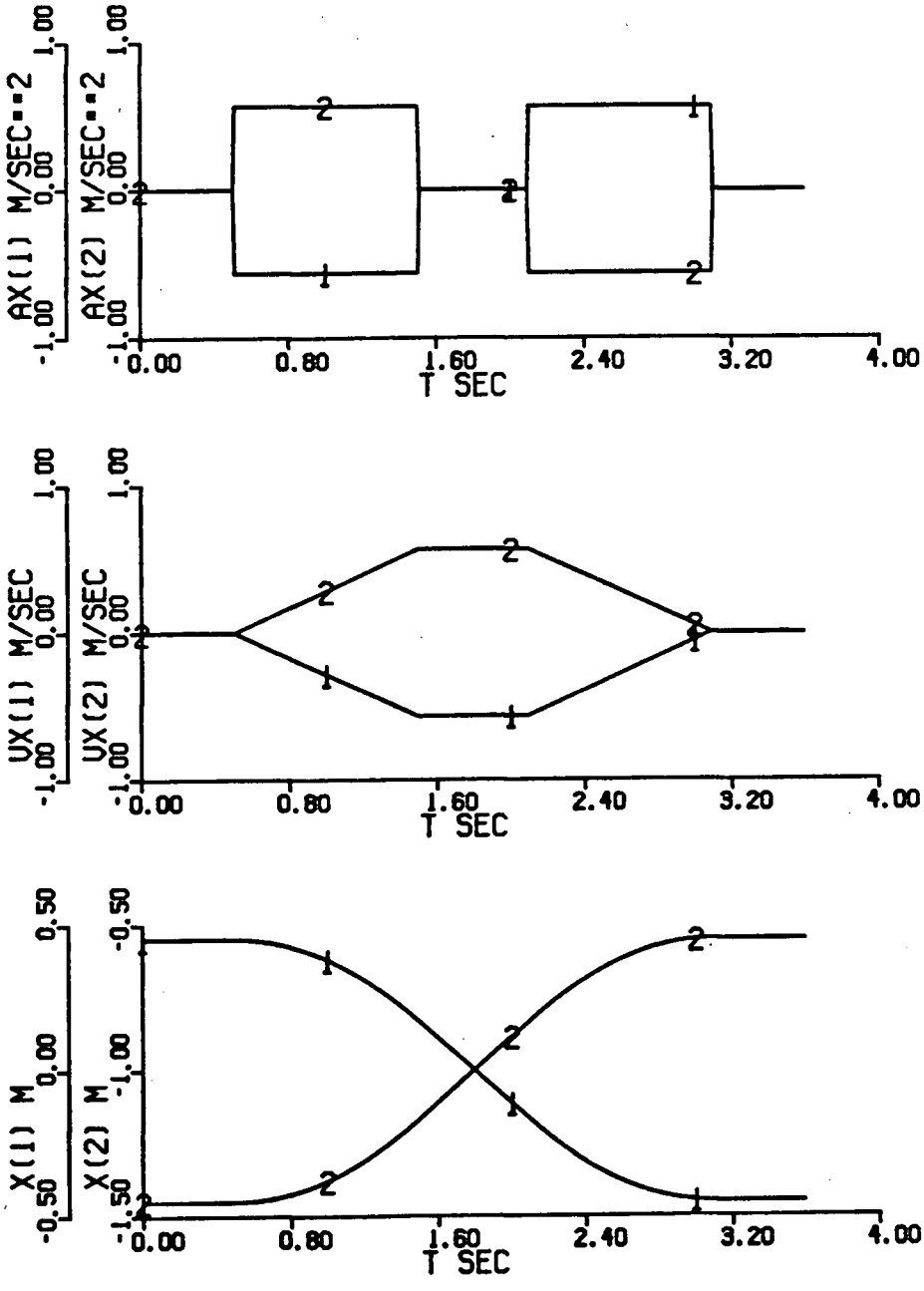


Figure 3.8 Ideal open loop control of line 1

to finite precision in the calculation of the analytical inverse kinematics and analytical inverse dynamics, inaccuracy in the simulation of the manipulator, finite precision in the calculation of the analytical direct kinematics, and the fact that torque is not updated continuously, albeit that torque is updated very frequently.

PATH	XERMS	VXERMS	AXERMS
line 1	2.54E-5	3.05E-5	1.05E-4
circle 1	2.10E-5	3.22E-5	7.78E-5

Table 3.3 Path error using ideal open loop control

While such ideal open loop control is not a realistic control strategy, it serves to test two aspects of the simulation program. First, it confirms that path specification is being performed correctly. The desired acceleration and velocity are indeed the derivatives of the desired velocity and position, respectively. Secondly, it confirms that the inverse kinematics and inverse dynamics are indeed the inverse of the direct dynamics and direct kinematics used. While this does not mean that the kinematics and dynamics are necessarily correct, it does make it very probable that, given that the analytical kinematics and analytical dynamics have been derived correctly, errors have not been introduced in the implementation of the simulation. If either of these assertions were not true, it

would be highly unlikely that the resulting path would duplicate the desired path so closely when using open loop control.

In addition, ideal open loop control allows one to examine the torques that must be applied to the manipulator to drive it through a specified path. Figure 3.9 shows the torque, as function of time, that is required to drive the manipulator through standard path, line 1.

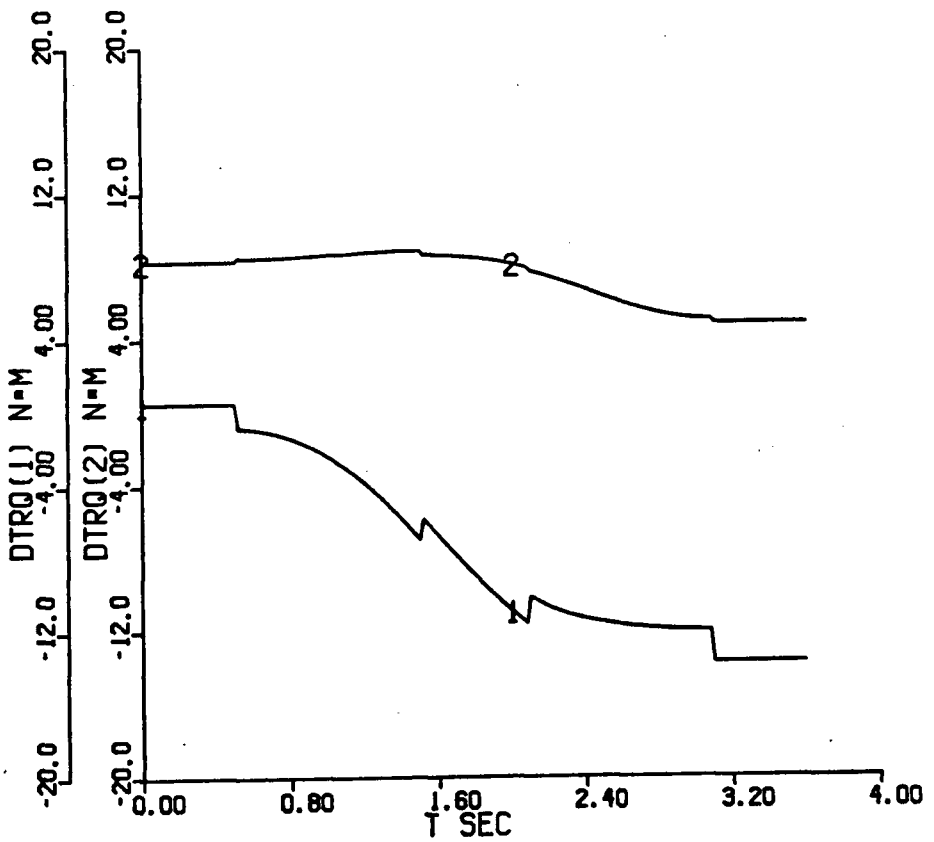


Figure 3.9 Torque profile for line 1

3.5.2 IDEAL CLOSED LOOP CONTROL

The position and velocity tracking errors that occur when using ideal open loop control can be reduced by using error correcting feedback resulting in ideal closed loop control. Ideal closed loop control assumes exact measurement of the the resulting Cartesian path and continuous updating of the applied torque. To test ideal closed loop control, two simulations were carried out with CLOSED=.TRUE., INVARM=.TRUE., EXACT=.TRUE., PRDICT=.FALSE., DELAY=.FALSE., k_1=16, k_2=64 and t_{calc}=0.0001 sec. The resulting path errors are shown in table 3.4 and are reduced from those occurring with ideal open loop control. In fact, any further reduction of path error is limited by the precision of calculations in the simulation.

The error correcting action of closed loop control is illustrated by a simulation in which the manipulator is perturbed from the desired position of $\langle x_{d1}, x_{d2} \rangle = \langle 0, -1 \rangle$ at time t=0.2 sec and is restored to the desired position by error correcting feedback. Such a simulation is shown in figure 3.10. In theory it would seem that one could achieve critical damping of such position and velocity errors with as short a time constant as desired simply by increasing k_1 and k_2 while maintaining the condition $k_1^2 = 4k_2$. In practice this cannot be done as ideal closed loop control cannot be achieved.

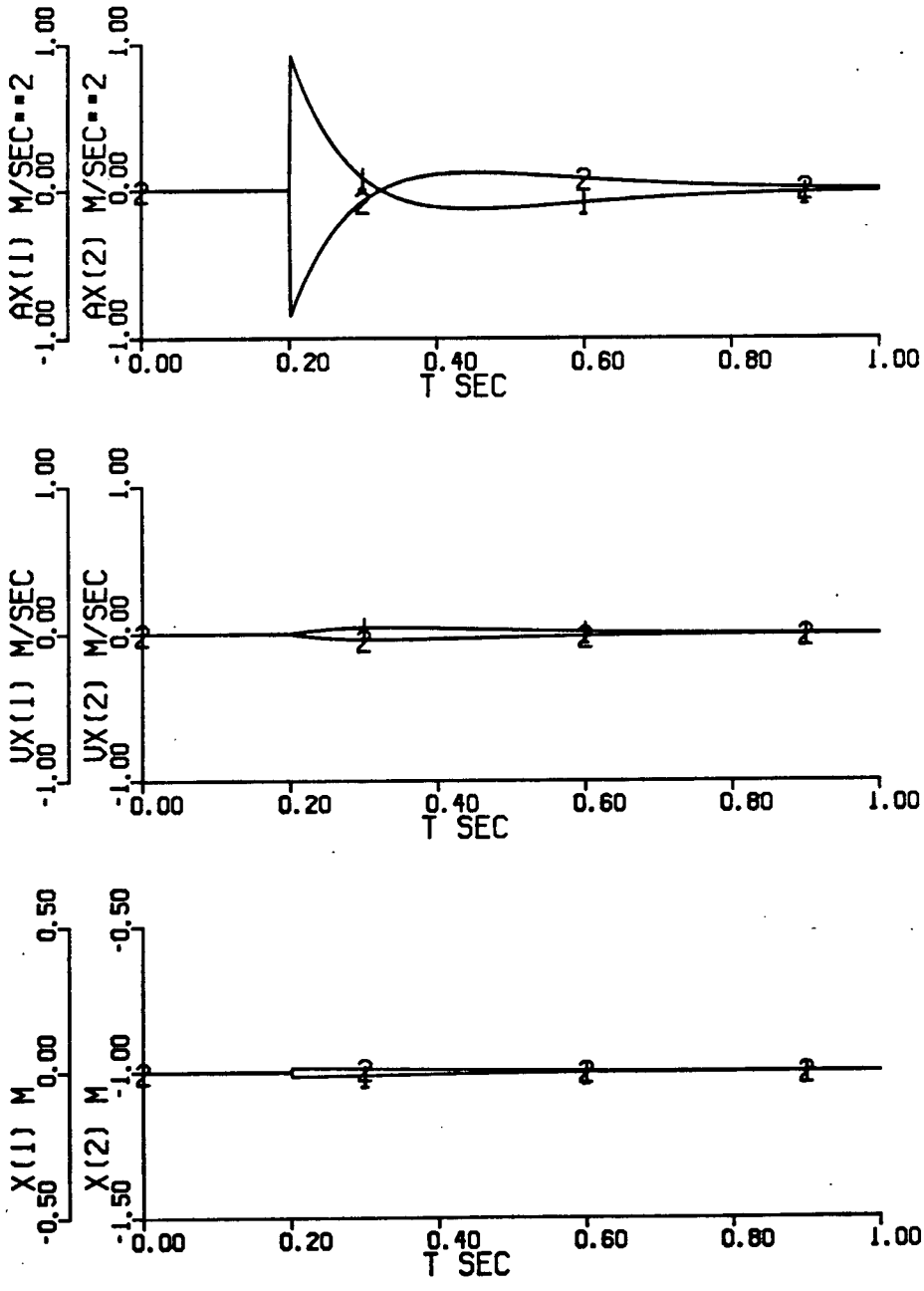


Figure 3.10 Error correcting action of ideal closed loop control

PATH	XERMS	VXERMS	AXERMS
line 1	2.37E-6	6.34E-6	1.46E-4
circle 1	1.56E-6	4.97E-6	1.01E-4

Table 3.4 Path error using ideal closed loop control

3.5.3 REALIZABLE CLOSED LOOP CONTROL - STANDARD ANALYTICAL CONTROL

In a practical implementation of closed loop control, various constraints necessitate deviation from ideality. These contraints typically have their roots in technological limitations on the speed at which calculations can be performed.

CALCULATION DELAY

A certain amount of time passes between the observation of the manipulator position, and the calculation of the torque to apply to the manipulator to correct position and velocity errors while tracking a path specification. There is thus a delay between observation of manipulator motion and error correcting action. Such a delay is implemented in the simulation when logical variable DELAY is true as follows,

$$\bar{x}_o(t) = \bar{x}_{se}(t-t_{calc}) \qquad (3.93)$$

Unless one uses parallel processing or pipelined processing, the applied torque will be quantized in time by this same calculation interval. To keep computer cost to a minimum, it is desirable to permit this calculation delay to be as large as possible without resulting in poor control. We have chosen

t_{calc}=0.01 sec as a reasonable compromise.

INEXACT VISION

Exact measurement of the manipulator path for use in error correcting feedback is an unrealistic assumption. A more realistic assumption, although still rather optimistic, is that the vision system estimates only position, $\bar{x}_v$, so accurately as to be considered equal to the exact position, $\bar{x}$. Velocity and acceleration would then be obtained by simple differentiation of the observed position as a function of time. We thus model the vision system as follows,

$$\bar{x}_e(t) = \bar{x}_v(t) = \bar{x}(t) \tag{3.94}$$

$$\dot{\bar{x}}_e(t) = [\bar{x}_e(t) - \bar{x}_e(t-t_{calc})]/t_{calc} \tag{3.95}$$

$$\ddot{\bar{x}}_e(t) = [\dot{\bar{x}}_e(t) - \dot{\bar{x}}_e(t-t_{calc})]/t_{calc} \tag{3.96}$$

Note that in addition to the previously mentioned delay of t_{calc}, there are additional effective delays of $t_{calc}/2$ for velocity observations and t_{calc} for acceleration observations.

PREDICTION

It is possible to reduce the effect of feedback delay by using prediction. Since one knows the desired velocity and acceleration when the observations are made, one can use this information to predict what the observed position and velocity should be at time t_{calc} later. These predictions can then be used in the calculation of the torque to apply at time t_{calc} later. To this end, we make use of position and velocity prediction as follows,

$$\bar{x}_p(t+t_{calc}) = \bar{x}_o(t) + \dot{\bar{x}}_d(t)t_{calc} + \ddot{\bar{x}}_d(t)t_{calc}^2/2 \tag{3.97}$$

$$\dot{\bar{x}}_p(t+t_{calc}) = \dot{\bar{x}}_o(t) + \ddot{\bar{x}}_d(t)t_{calc} \tag{3.98}$$

Note that it is also possible to use the observed velocity and acceleration for prediction purposes, rather than the desired velocity and acceleration. Such use of observations, however, was found to have certain disadvantages and was not adopted.

FEEDBACK GAIN

The feedback gains, k_1 and k_2, cannot be made arbitrarily large in practice. A choice of k_1=16 and k_2=64 was found to result in good error correction with a minimal tendency to cause instability given our choice of parameters for standard analytical control; closed loop control using the analytical inverse kinematics, analytical inverse dynamics and realistic constraints.

STANDARD ANALYTICAL CONTROL

Standard analytical control is simulated by setting CLOSED=.TRUE., INVARM=.TRUE., EXACT=.FALSE., VISION=.TRUE, DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.FALSE., k_1=16, k_2=64 and t_{calc}=0.01 sec. To test standard analytical control, simulations were done of the six standard paths. The resulting path errors are shown in table 3.5. Figure 3.11 shows the resulting path in Cartesian coordinates for the simulation of line 1. Position and velocity tracking is very good but there is a slight overshooting of steps in acceleration. Tracking is significantly poorer, though, than that which appeared possible using ideal open loop or ideal closed loop control. The performance of standard analytical control shown here forms a benchmark for the evaluation of learned control. One should hope to achieve similar results with learned control, but should not expect to

exceed the performance of standard analytical control.

PATH	XERMS	VXERMS	AXERMS
line 1	5.99E-4	2.29E-3	9.54E-2
line 2	7.09E-4	2.44E-3	9.58E-2
line 3	6.75E-4	2.45E-3	9.92E-2
line 4	6.32E-4	2.42E-3	9.87E-2
circle 1	6.53E-4	1.50E-3	4.73E-2
circle 2	6.18E-4	1.48E-3	4.73E-2

Table 3.5 Path error using standard analytical control

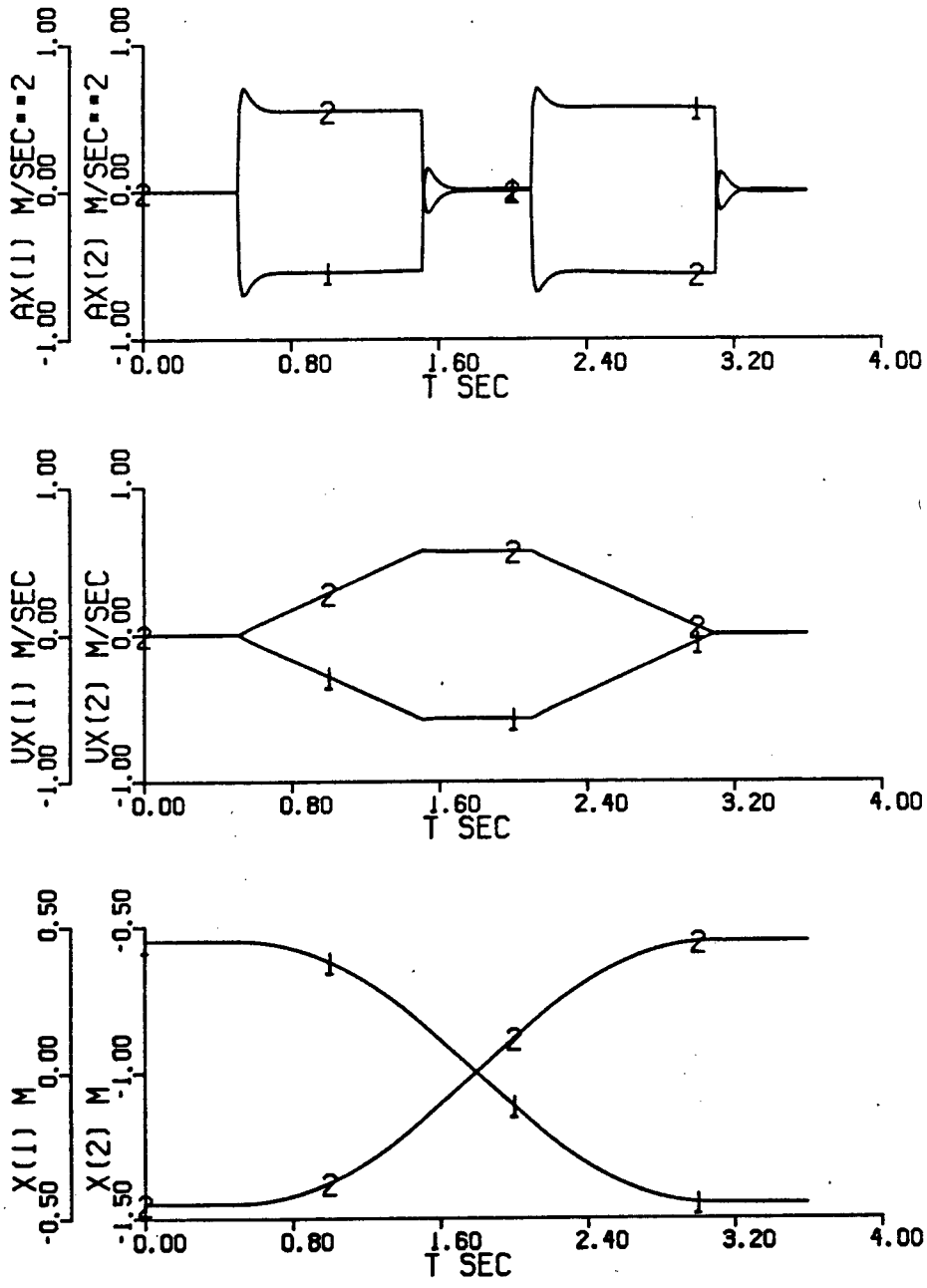


Figure 3.11 Standard analytical control of line 1

3.5.4 CHOICE OF PARAMETERS DEFINING STANDARD ANALYTICAL CONTROL

The parameters used in standard analytical control were chosen so as to achieve good control with the constraint that the system be realizable. The appropriateness of the chosen parameters can be demonstrated by simulations using standard analytical control except that individual parameters are varied.

CALCULATION DELAY

Calculation delay is unavoidable in a real implementation. It was chosen to be as large as possible without resulting in poor control, as this permits implementation with lower cost computing hardware. Figures 3.12, 3.13 and 3.14 show simulations of line 1 for the cases t_{calc}=0.02 sec, t_{calc}=0.04 sec and t_{calc}=0.04 sec, respectively. The path error for these simulations is shown in table 3.6. It can be seen that as t_{calc} is increased from 0.01 sec, the standard value, the control becomes progressively poorer. For the case t_{calc}=0.02 sec, position and velocity tracking is quite good, however, there is a noticeable oscillation in the acceleration of the manipulator. This could cause excessive vibration of the links of an actual manipulator, an effect that is not modelled here. It is assumed that the torque must be updated at a reasonably high rate to avoid resonance with the vibrational modes of manipulator links, otherwise manipulators would have to be specially constructed to damp vibrations.

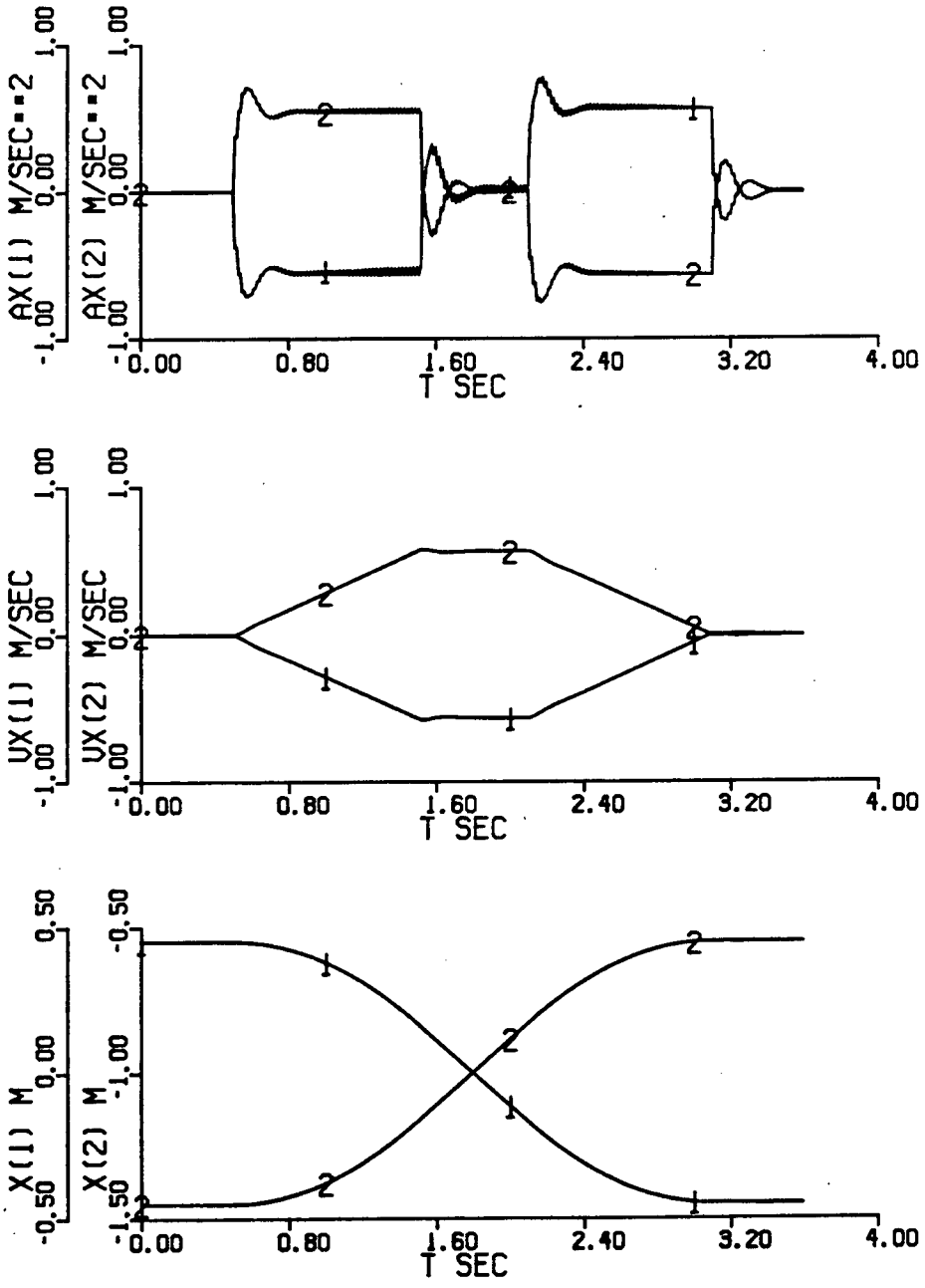


Figure 3.12 Standard analytical control of line 1, except that t_{calc}=0.02 sec

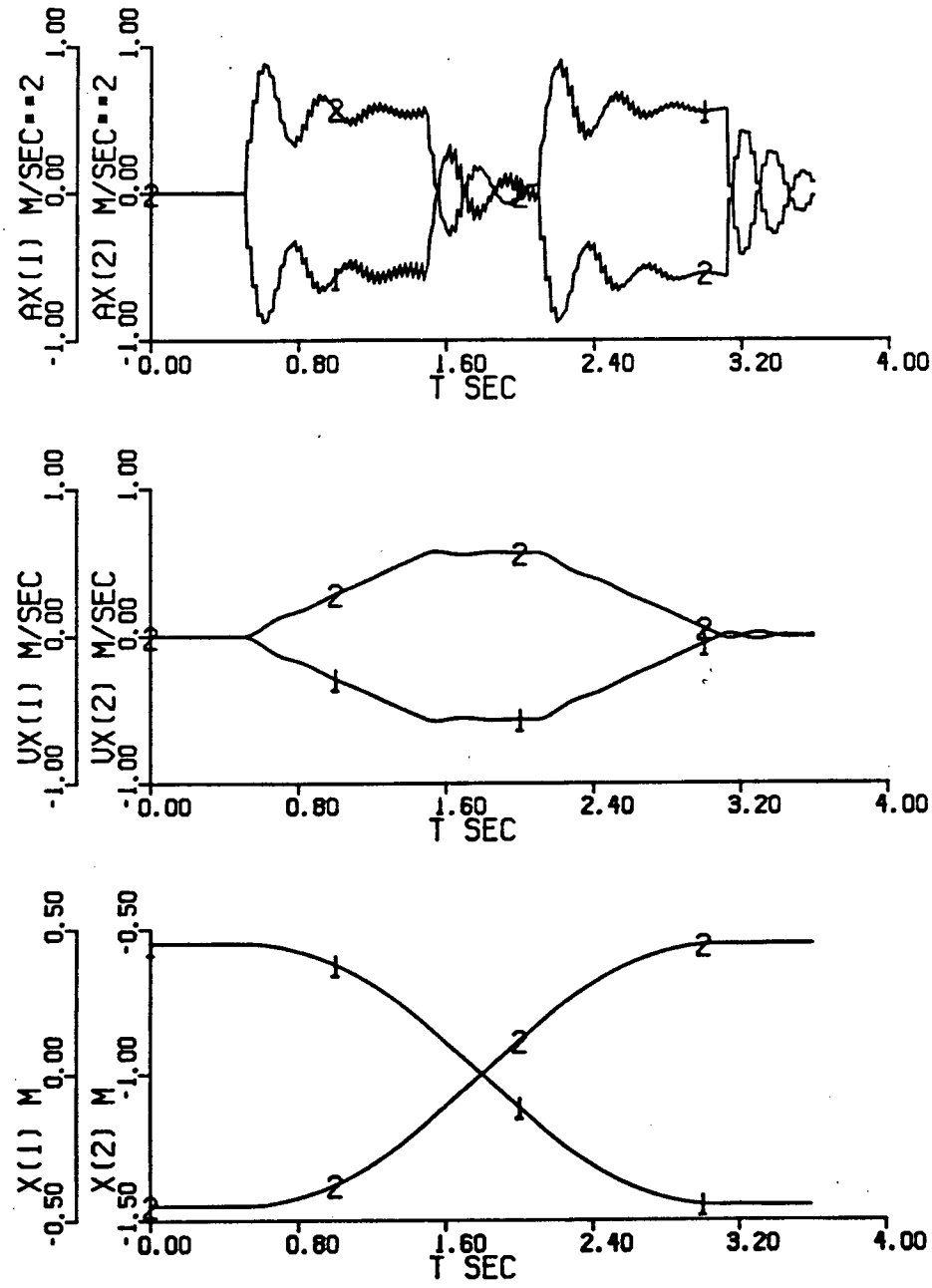


Figure 3.13 Standard analytical control of line 1, except that t_{calc}=0.03 sec

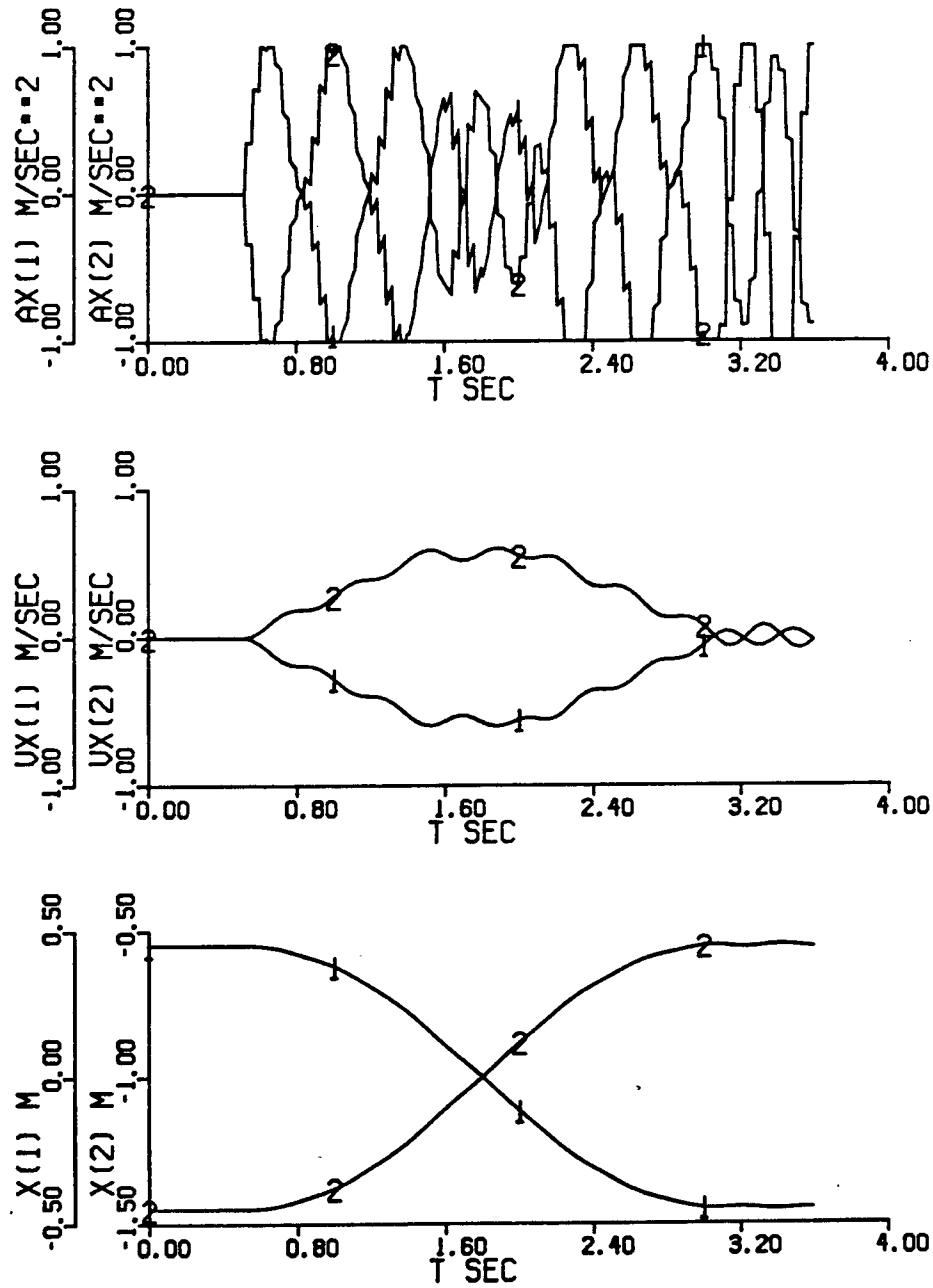


Figure 3.14 Standard analytical control of line 1, except that t_{calc}=0.04 sec

INEXACT VISION

Accurate measurement of manipulator motion is necessary for feedback to be useful in improving control. Errors in measurement translate directly into tracking errors. It is thus necessary to invest in a very accurate vision system, or its equivalent using the analytical direct kinematics, if good control is to be achieved. It does seem reasonable, though, that only Cartesian position needs to be measured accurately. Cartesian velocity and acceleration, if required, can be adequately obtained by differentiating the position. Figure 3.15 shows a simulation of line 1 in which exact vision is used for feedback purposes by setting EXACT=.TRUE. The resulting path error is shown in table 3.6. It can be seen that control with exact vision is not significantly better than with our model of a vision system where velocity and acceleration measurements are obtained by differentiating accurate measurements of position.

PREDICTION

Prediction has been used in standard analytical control. The benefits of prediction are shown by a simulation in which no prediction was used. The resulting path is shown in figure 3.16 and the path error is listed in table 3.6. Lack of prediction results in more pronounced overshooting of steps in acceleration and a significant increase in path error as compared to results using prediction. For the two link manipulator, prediction requires only 6 multiplications and 6 additions. Prediction was thus adopted for use in standard analytical control as it requires a minimal amount of additional computation and

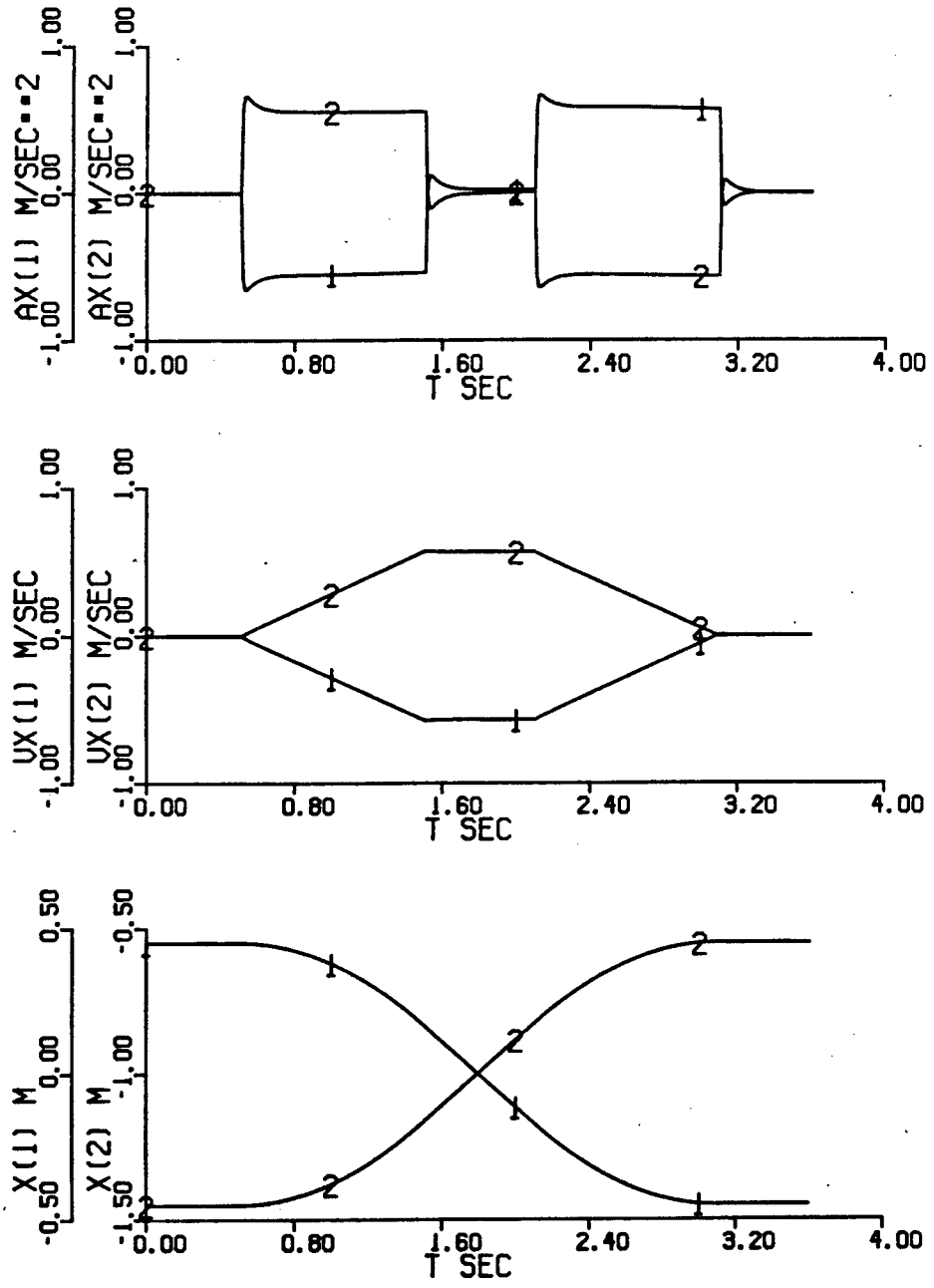


Figure 3.15 Standard analytical control of line 1, except that EXACT=.TRUE.

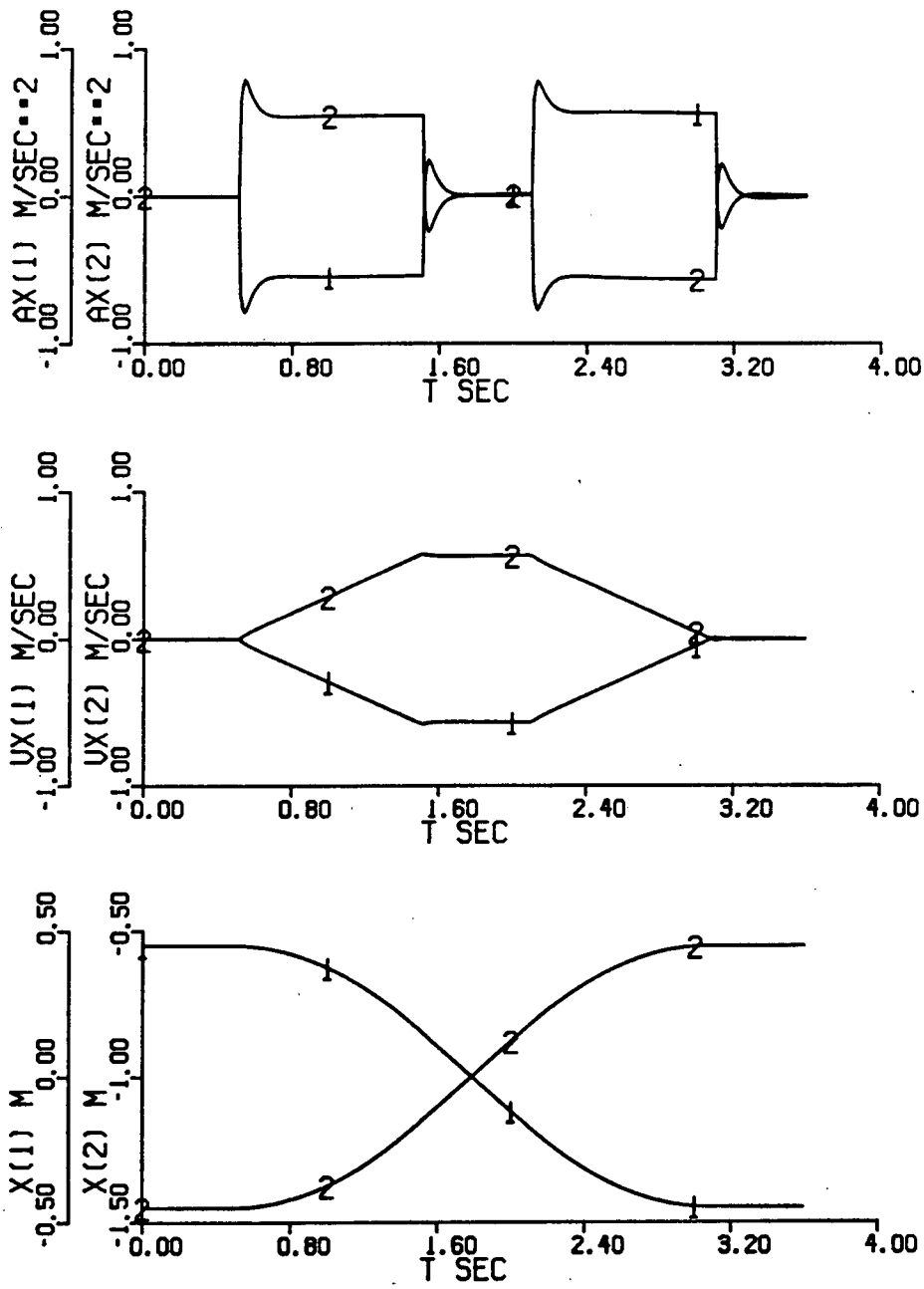


Figure 3.16 Standard analytical control of line 1, except that PRDICT=.FALSE.

significantly improves tracking.

In standard analytical control, prediction is implemented using the desired velocity and acceleration. It is also possible to use the observed velocity and acceleration for this purpose. Figure 3.17 shows a simulation of line 1 with USEOBS=.TRUE. The resulting path error is shown in table 3.6. Tracking is no better than with prediction using the desired velocity and acceleration specification. When control is marginal such that the resulting velocity and acceleration differ somewhat from the desired velocity and acceleration, it might be more beneficial to use the observed velocity and acceleration for prediction. We see prediction, however, as a way of improving control that is already adequate, not as a way of making inadequate control become adequate. Also, observation of acceleration is not otherwise required, except when learning, as will be shown later. Observation of acceleration is thus an additional computational expense with little benefit. Hence we adopted prediction using the desired velocity and acceleration for use in standard analytical control.

FEEDBACK GAIN

With the departures from ideality present in realizable, closed loop control, it is not possible to arbitrarily increase the feedback gains, k_1 and k_2, to achieve ever better control. Because of these departures from ideality, the non-linearities of the manipulator and vision system are only approximately cancelled by the analytical inverse dynamics and analytical inverse kinematics. Thus the characterization of the control

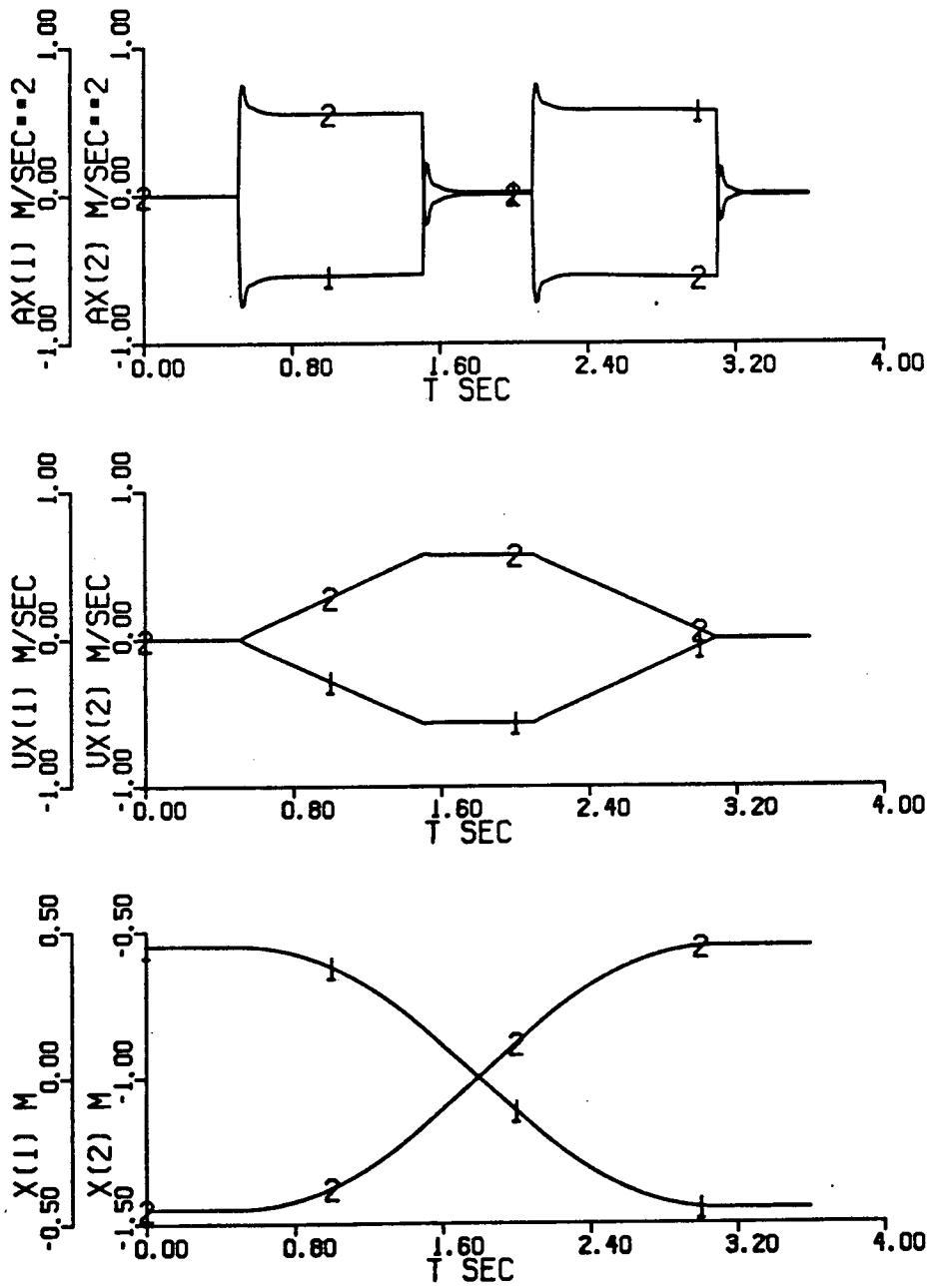


Figure 3.17 Standard analytical control of line 1, except that USEOBS=.TRUE.

system by a second order, linear, homogeneous differential equation with constant coefficients as in (3.31) is only approximately true. We found that k_1=16 and k_2=64 gave about the best control in combination with the other chosen parameters defining standard analytical control. Figure 3.18 shows a simulation of line 1 with k_1=64 and k_2=1024. The control system is clearly becoming oscillatory as the feedback gains are increased. Figure 3.19 shows a simulation of line 1 where the feedback gains are reduced to k_1=4 and k_2=4. Now the control system is not oscillatory but the path error is increased from that of standard analytical control due to the slower correction of velocity and position errors. The effect of departure from the constraint that $k_1^2=4k_2$ is shown in figures 3.20 and 3.21 where line 1 is simulated with feedback gains of k_1=8, k_2=128 and k_1=32, k_2=32, respectively. The path errors for these simulations with various feedback gains are listed in table 3.6. It can be seen that k_1=16 and k_2=64 are reasonable choices for use in standard analytical control.

Standard analytical control has been established by reasonably choosing the parameters necessary to implement a realizable, closed loop control system for the two link manipulator. Standard analytical control thus serves as a reasonable benchmark against which to compare learned control.

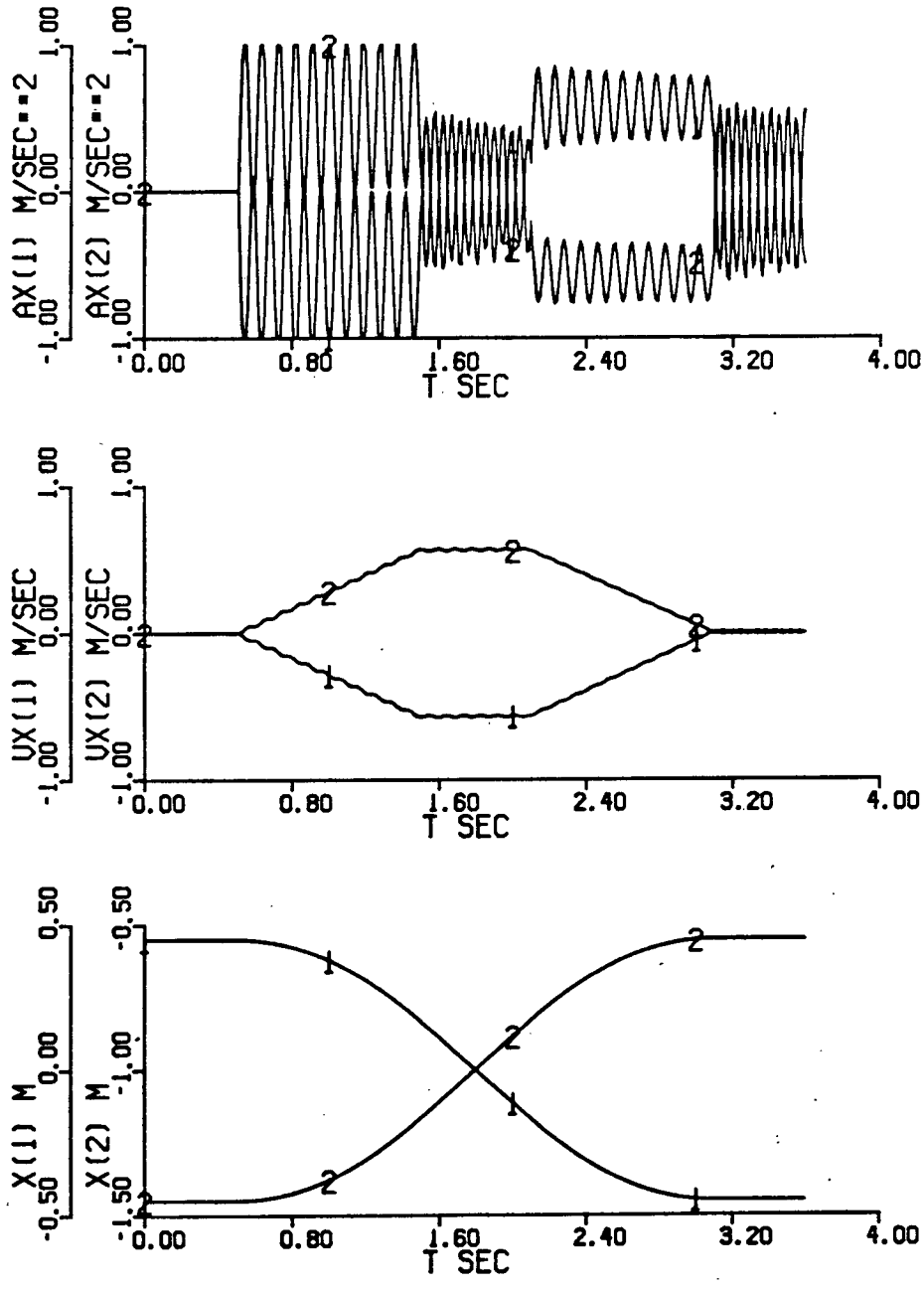


Figure 3.18 Standard analytical control of line 1, except that k_1=64 and k_2=1024

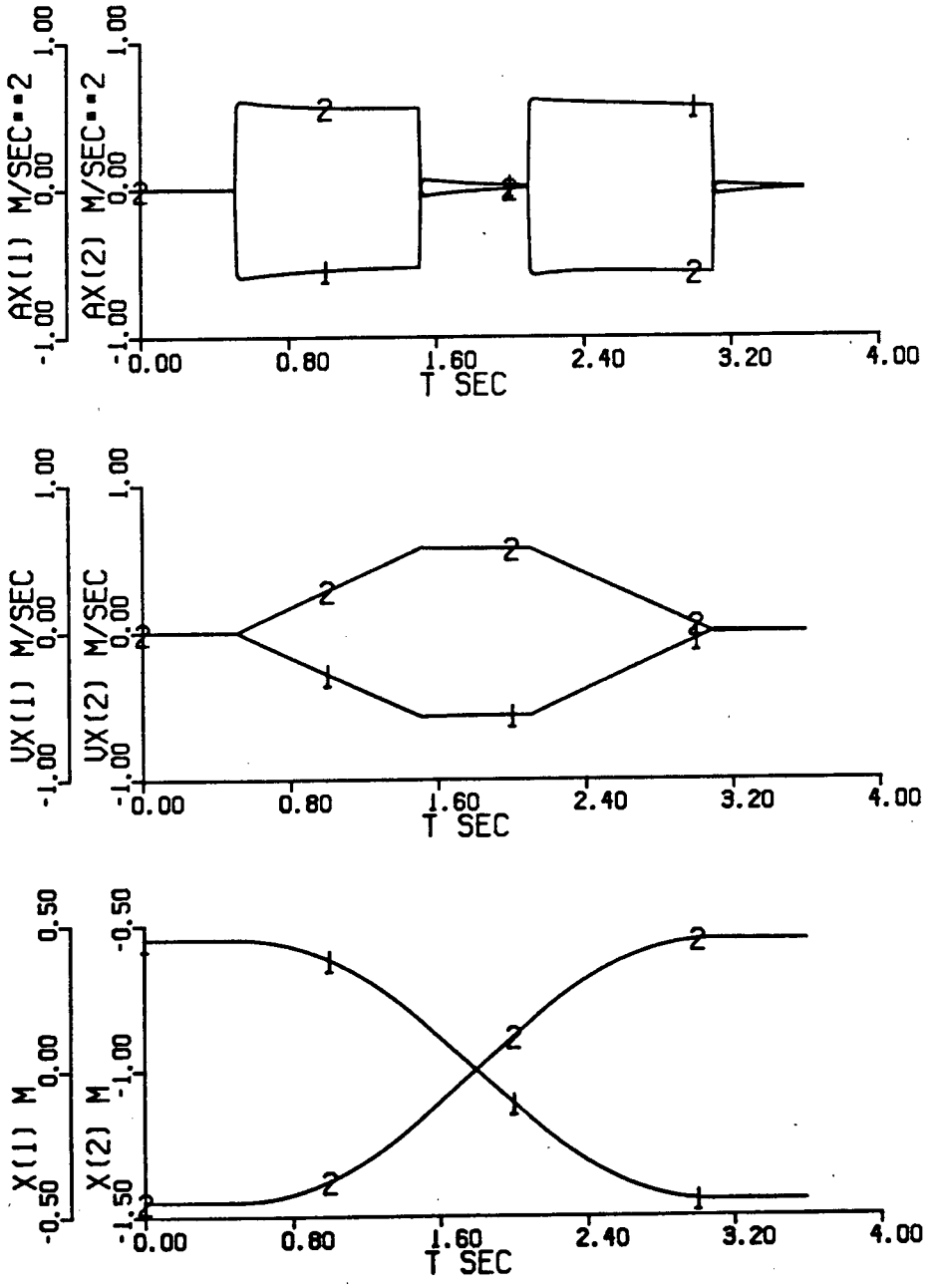


Figure 3.19 Standard analytical control of line 1, except that $k_1=4$ and $k_2=4$

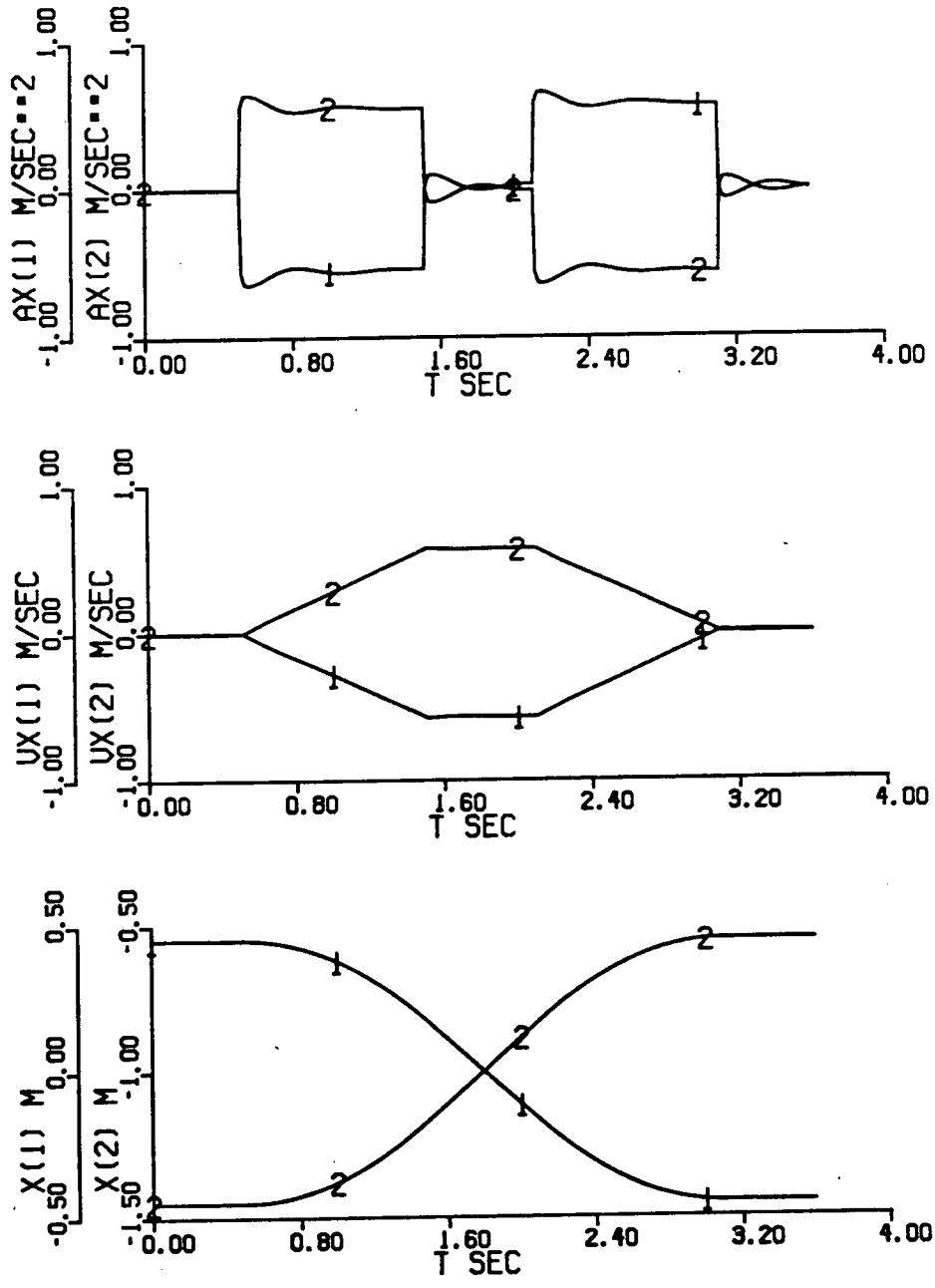


Figure 3.20 Standard analytical control of line 1, except that k_1=8 and k_2=128

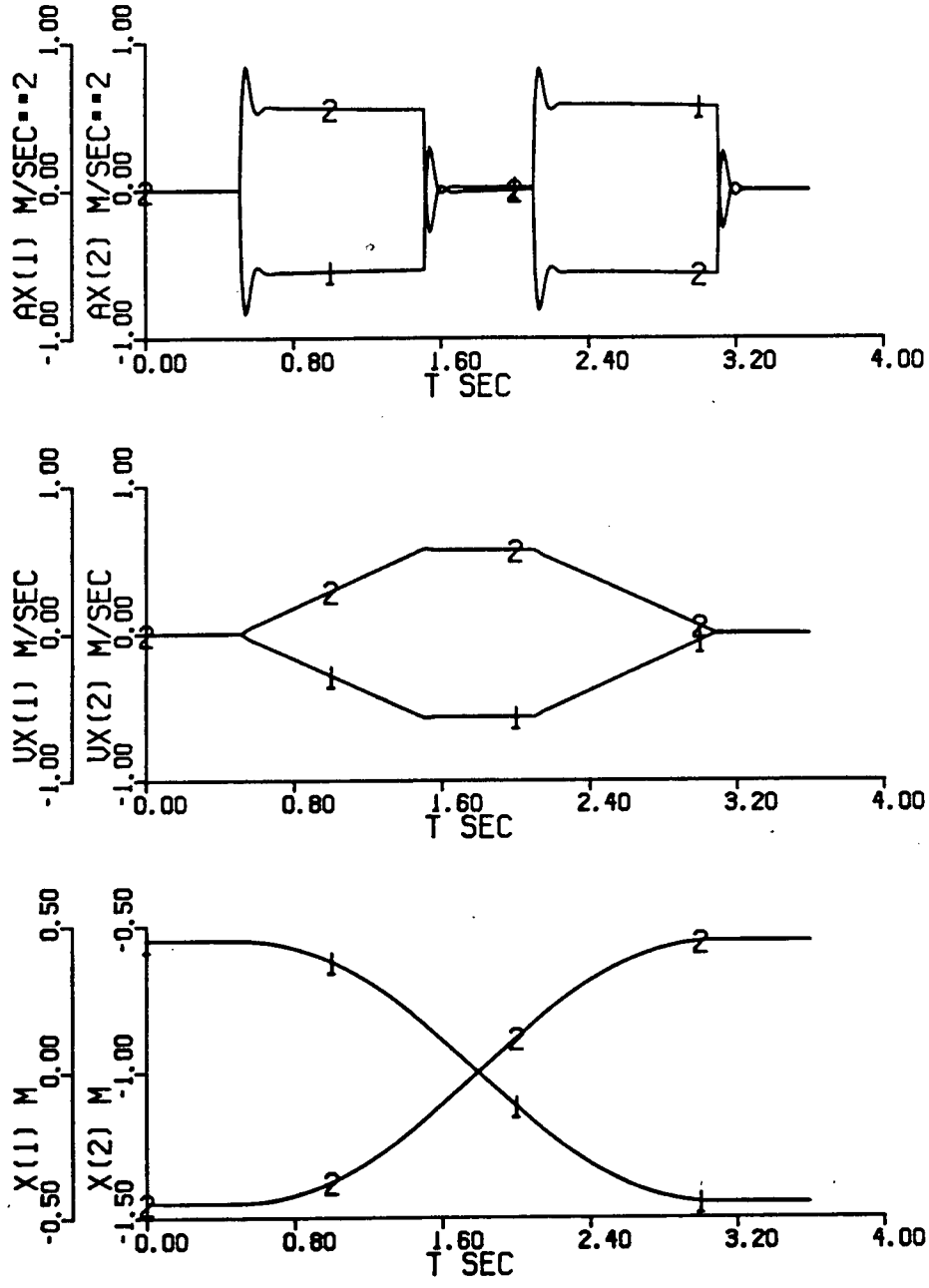


Figure 3.21 Standard analytical control of line 1, except that k_1=32 and k_2=32

PATH	XERMS	VXERMS	AXERMS	NOTES
line 1	5.99E-4	2.29E-3	9.54E-2	Standard
line 1	1.12E-3	4.53E-3	1.20E-1	t_{calc} = 0.02 sec
line 1	1.62E-3	9.70E-3	2.06E-1	t_{calc} = 0.03 sec
line 1	2.87E-3	3.71E-2	6.45E-1	t_{calc} = 0.04 sec
line 1	2.60E-4	1.64E-3	9.02E-2	Exact Vision
line 1	4.57E-3	7.06E-3	1.12E-1	No Prediction
line 1	6.60E-4	2.05E-3	9.66E-2	Use Observations
line 1	1.55E-4	6.57E-3	4.38E-1	k_1 = 64, k_2 = 1024
line 1	2.44E-3	3.06E-3	8.74E-2	k_1 = 4, k_2 = 4
line 1	2.47E-4	2.64E-3	9.35E-2	k_1 = 8, k_2 = 128
line 1	1.36E-3	2.92E-3	1.09E-1	k_1 = 32, k_2 = 32

Table 3.6 Path error using standard analytical control, except for noted variations

3.6 ADEQUACY OF A SUM OF POLYNOMIALS REPRESENTATION OF THE INVERSE DYNAMICS AND INVERSE KINEMATICS

A principal goal of this work is to learn a sum of polynomials representation of the inverse dynamics and inverse kinematics of the two link manipulator without recourse to analysis of the manipulator. Before considering how such learning is to be performed, it is instructive to consider whether the inverse dynamics and inverse kinematics can even be adequately represented by a sum of polynomials for the purposes of control.

3.6.1 DERIVATION OF A SUM OF POLYNOMIALS REPRESENTATION OF THE INVERSE DYNAMICS

Although the inverse dynamics are quite complex, it is straightforward to derive a sum of polynomials representation that closely approximates the analytical inverse dynamics.

The analytical inverse dynamics are given by equations (3.51) through (3.66). To derive a sum of polynomials equivalent it is necessary to consider a two link manipulator with specific parameters. To this end we have chosen a two link manipulator as follows,

$$m_1 = 1 \text{ Kg} \tag{3.99}$$

$$m_2 = 1 \text{ Kg} \tag{3.100}$$

$$l_1 = 1 \text{ m} \tag{3.101}$$

$$l_2 = 1 \text{ m} \tag{3.102}$$

The standard gravitational acceleration has been assumed;

$$g = 9.81 \text{ m/sec}^2 \tag{3.103}$$

For these choices of parameters, the coefficients in the

analytical inverse dynamics are, neglecting units, as follows,

$$d_{11} = 3 + 2\cos(a_2) \tag{3.104}$$

$$d_{12} = 1 + \cos(a_2) \tag{3.105}$$

$$d_{111} = 0 \tag{3.106}$$

$$d_{122} = -\sin(a_2) \tag{3.107}$$

$$d_{112} = -\sin(a_2) \tag{3.108}$$

$$d_{121} = -\sin(a_2) \tag{3.109}$$

$$d_1 = 19.62\sin(a_1) + 9.81\sin(a_1+a_2) \tag{3.110}$$

and,

$$d_{21} = 1 + \cos(a_2) \tag{3.111}$$

$$d_{22} = 1 \tag{3.112}$$

$$d_{211} = \sin(a_2) \tag{3.113}$$

$$d_{222} = 0 \tag{3.114}$$

$$d_{212} = 0 \tag{3.115}$$

$$d_{221} = 0 \tag{3.116}$$

$$d_2 = 9.81\sin(a_1+a_2) \tag{3.117}$$

With these coefficients the analytical inverse dynamics are,

$$\begin{aligned}\tau_1 = {} & 3\ddot{a}_1 + 2\cos(a_2)\ddot{a}_1 + \ddot{a}_2 + \cos(a_2)\ddot{a}_2 \\ & - \sin(a_2)\dot{a}_2^{\,2} - 2\sin(a_2)\dot{a}_1\dot{a}_2 \\ & + 19.62\sin(a_1) + 9.81\sin(a_1+a_2)\end{aligned} \tag{3.118}$$

$$\begin{aligned}\tau_2 = {} & \ddot{a}_1 + \cos(a_2)\ddot{a}_1 + \ddot{a}_2 \\ & + \sin(a_2)\dot{a}_1^{\,2} + 9.81\sin(a_1+a_2)\end{aligned} \tag{3.119}$$

The sine and cosine terms can be approximated over the range $-\pi < a_i < \pi$ using their equivalent series representations, truncated to 4^{th} order and lower terms,

$$\cos(a_2) \simeq 1 - a_2^{\,2}/2 + a_2^{\,4}/24 \tag{3.120}$$

$$\sin(a_2) \simeq a_2 - a_2^{\,3}/6 \tag{3.121}$$

$$\sin(a_1) \simeq a_1 - a_1^3/6 \tag{3.122}$$

$$\sin(a_1+a_2) \simeq a_1 + a_2 - a_1^3/6 - a_1^2a_2/2 - a_1a_2^2/2 - a_2^3/6 \tag{3.123}$$

Substituting these approximations for the sine and cosine terms and keeping the resulting terms of 4^{th} order or lower yields a sum of polynomials representation for the inverse dynamics,

$$\begin{aligned}\tau_1 \simeq\ & 29.4a_1 - 4.91a_1^3 + 9.81a_2 - 4.91a_1^2a_2 - 4.91a_1a_2^2 \\ & - 1.64a_2^3 - 2a_2\dot{a}_1\dot{a}_2 - a_2\dot{a}_2^2 + 5\ddot{a}_1 - a_2^2\ddot{a}_1 \\ & + 2\ddot{a}_2 - 0.5a_2^2\ddot{a}_2\end{aligned} \tag{3.124}$$

$$\begin{aligned}\tau_2 \simeq\ & 9.81a_1 - 1.64a_1^3 + 9.81a_2 - 4.91a_1^2a_2 - 4.91a_1a_2^2 \\ & - 1.64a_2^3 + a_2\dot{a}_1^2 + 2\ddot{a}_1 - 0.5a_2^2\ddot{a}_1 + \ddot{a}_2\end{aligned} \tag{3.125}$$

Demonstration of the accuracy achievable with such a 4^{th} order sum of polynomials representation is done in the following sections.

3.6.2 PRE-LEARNING OF THE INVERSE DYNAMICS

Having derived a sum of polynomials representation of the inverse dynamics, it seemed natural to determine whether this representation could be learned using the analytical inverse dynamics as a guide. We call this pre-learning as the sum of polynomials representation is learned off-line, before being used in any control scheme. This was done using a program analagous to that used in chapter 2 to investigate the convergence rates of the various learning algorithms except that the target functions used were the two functions that make up the inverse dynamics. Pre-learning was carried out using Method 2, Learning Identification, for the cases, $\overset{*}{f} = \tau_1$ and $\overset{*}{f} = \tau_2$. The number of input variables was v = 6 and the system order was

s = 4. The input variables were,

$$z_1 = a_1 \tag{3.126}$$

$$z_2 = a_2 \tag{3.127}$$

$$z_3 = \dot{a}_1 \tag{3.128}$$

$$z_4 = \dot{a}_2 \tag{3.129}$$

$$z_5 = \ddot{a}_1 \tag{3.130}$$

$$z_6 = \ddot{a}_2 \tag{3.131}$$

Training points were randomly generated uniformly over the space, $-1 < z_i < 1$, and 20,000 iterations were performed. Torques τ_1 and τ_2 were learned as sums of polynomials of the input variables, z_i.

Figure 3.22 shows the reduction of Δf_1 and Δf_2 as a function of the number of training iterations. The graphs show the magnitudes of Δf_1 and Δf_2, averaged over intervals of 100 training iterations. It can be seen that the average error in estimating the target functions (in this case the torques τ_1 and τ_2) has dropped to about 0.05 N·m. Note that the estimates of the torques do not improve after about 5000 iterations. This implies that better accuracy cannot be achieved without increasing, s, the system order.

Table 3.7 and 3.8 shows the coefficients derived previously and shown in equations (3.124) and (3.125) along with their counterparts that were pre-learned. Clearly, as learning proceeds, the coefficients are converging to values close to those obtained by derivation. One should not expect an exact correspondence as the best 4^{th} order approximation of the analytical inverse dynamics is not likely to be exactly that

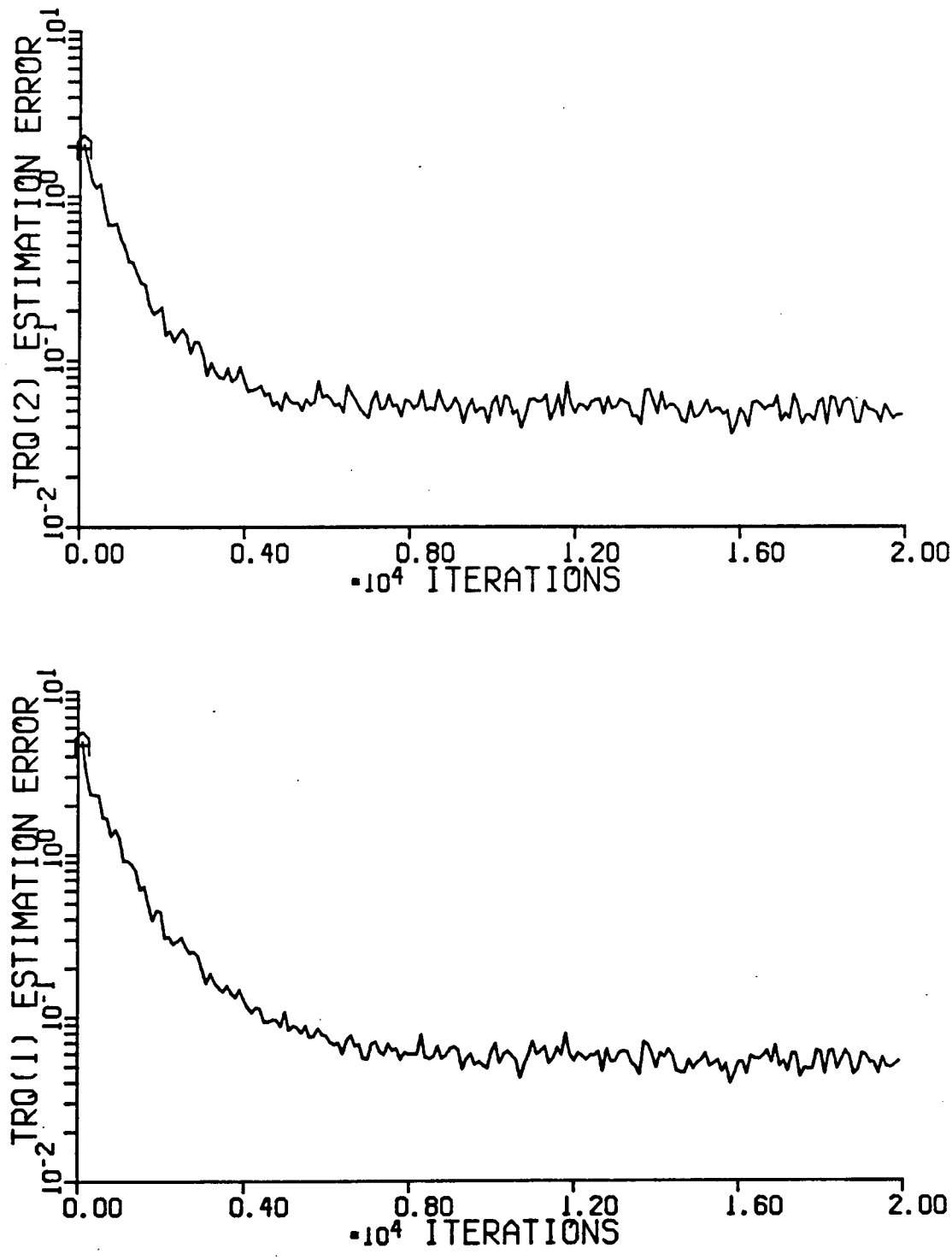


Figure 3.22 Reduction of estimation error during pre-learning of the inverse dynamics (estimation error averaged over each 100 iteration interval)

obtained by truncating the series representations of the sine and cosine terms, as done here. Furthermore, Learning Identification does not quite converge to the best 4^{th} order approximation as it is limited by the constraint that learning eliminates estimation error at the last training point. There should be, however, a close correspondence between the derived and pre-learned coefficients.

POLYNOMIAL TERM	DERIVED COEFFICIENT	PRE-LEARNED COEFFICIENT
z_1	29.4	29.3
z_1^3	-4.91	-4.47
z_2	9.81	9.67
$z_2 z_1^2$	-4.91	-4.24
$z_2^2 z_1$	-4.91	-4.23
z_2^3	-1.64	-1.33
$z_4 z_3 z_2$	-2.00	-1.81
$z_4^2 z_2$	-1.00	-0.92
z_5	5.00	5.00
$z_5 z_2^2$	-1.00	-0.91
z_6	2.00	1.99
$z_6 z_2^2$	-0.50	-0.51
others	0.00	<0.06

Table 3.7 Coefficients for a sum of polynomials representation of the inverse dynamics function for torque τ_1 obtained by derivation and by pre-learning

POLYNOMIAL TERM	DERIVED COEFFICIENT	PRE-LEARNED COEFFICIENT
z_1	9.81	9.69
z_1^3	-1.64	-1.36
z_2	9.81	9.71
$z_2 z_1^2$	-4.91	-4.24
$z_2^2 z_1$	-4.91	-4.22
z_2^3	-1.64	-1.41
$z_3^2 z_2$	1.00	0.90
z_5	2.00	2.00
$z_5 z_2^2$	-0.50	-0.44
z_6	1.00	0.99
others	0.00	<0.05

Table 3.8 Coefficients for a sum of polynomials representation of the inverse dynamics function for torque τ_2 obtained by derivation and by pre-learning

3.6.3 ATTEMPTED DERIVATION OF A SUM OF POLYNOMIALS REPRESENTATION OF THE CARTESIAN INVERSE DYNAMICS

To achieve our goal of learned control with path specification in Cartesian coordinates, it is necessary to learn the Cartesian inverse dynamics. Thus it is instructive to attempt to derive a sum of polynomials representation of the inverse kinematics of the two link manipulator, which together with the inverse dynamics form the Cartesian inverse dynamics of the manipulator as given in (3.19). It was found that an

adequate sum of polynomials representation of the Cartesian inverse dynamics could only be achieved over a portion at a time of the space over which the manipulator is capable of moving.

The inverse kinematics of the two link manipulator are given by equations (3.39) through (3.44). Since it is the sines or cosines of the angular positions, rather than the angles themselves, that are required for substitution into the inverse dynamics, it is useful to consider equations (3.44) through (3.48) rather than the inverse kinematics functions for angular position.

Given the link lengths chosen previously, the formulas for the sines and cosines of angular position become,

$$\cos(a_2) = x_1^2/2 + x_2^2/2 - 1 \qquad (3.132)$$

$$\sin(a_2) = (1 - \cos^2(a_2))^{1/2} \qquad (3.133)$$

$$\cos(a_1) = x_1\sin(a_2)/(x_1^2+x_2^2) - x_2/2 \qquad (3.134)$$

$$\sin(a_1) = (1 - \cos^2(a_1))^{1/2} \qquad (3.135)$$

The first two functions given here, (3.132) and (3.133), can be well represented by a sum of polynomials over the whole space within the manipulator's reach, except very close to the origin. The second two functions, (3.134) and (3.135), appear to not be representable by a sum of polynomials due to the term $(x_1^2+x_2^2)$ in the denominator of (3.134).

The inverse kinematics functions for $\dot{a}_1$, $\dot{a}_2$, $\ddot{a}_1$ and $\ddot{a}_2$ all have the term $\sin(a_2)$ appearing in a denominator position. This means that these functions have singularities at the origin and at the limits of the manipulator's reach where $\sin(a_2)$ is zero.

The singularities at the origin and at the limits of the

manipulator's reach reflect control difficulties that are inherent with the manipulator. One should not expect to have good control at these points using path specification in Cartesian coordinates. One would normally not utilize the manipulator at these problematic points in the space.

The difficulty in representing $\cos(a_1)$ with a sum of polynomials is more significant. Even neglecting points near the origin and near the limits of the manipulator's reach, it does not appear to be possible to represent function (3.134) as a sum of polynomials over the rest of the manipulator's reach. It seems that any sum of polynomials that is good in one quadrant is bad in the opposite quadrant. Attempts to learn $\cos(a_1)$ over a circular band excluding the origin and the limits of the manipulator's reach proved futile. It was thus necessary to limit learning of the inverse kinematics and hence the Cartesian inverse dynamics to a portion of the space.

3.6.4 LIMITATION OF A SUM OF POLYNOMIALS REPRESENTATION OF THE CARTESIAN INVERSE DYNAMICS TO A PORTION OF THE MANIPULATOR'S SPACE

To permit a sum of polynomials representation of the Cartesian inverse dynamics, it was necessary to limit learning to a portion of the manipulator's space. This is not a great restriction of the technique as most real manipulators have revolute joints that are constrained to within a fraction of a complete circle of motion. One normally uses the manipulator in a restricted region or workspace, typically in front of or below the manipulator origin. Further, one can use several different

sum of polynomials representations of the Cartesian inverse dynamics, each for a different portion of the manipulator's space.

We restricted our attention to a workspace below the origin of the two link manipulator, namely the region specified by,

$$-0.5 < x_1 < 0.5 \text{ m} \tag{3.136}$$

$$-1.5 < x_2 < -0.5 \text{ m} \tag{3.137}$$

Velocity and acceleration were restricted to magnitudes of less than 1 m/sec and 1 m/sec^2, respectively. The normalized input variables for the learning algorithms were thus,

$$z_1 = 2x_1 \tag{3.138}$$

$$z_2 = 2x_2 + 2 \tag{3.139}$$

$$z_3 = \dot{x}_1 \tag{3.140}$$

$$z_4 = \dot{x}_2 \tag{3.141}$$

$$z_5 = \ddot{x}_1 \tag{3.142}$$

$$z_6 = \ddot{x}_2 \tag{3.143}$$

Over this portion of the manipulator's reach it is possible to represent the Cartesian inverse dynamics as sums of polynomials.

The cosine of a_2 is already exactly represented as a sum of polynomials of Cartesian coordinates,

$$\cos(a_2) = z_1^2/6 + z_2^2/6 - z_2/2 - 1/2 \tag{3.144}$$

The sine of a_2 is equal to the binomial expansion of the right hand side of (3.133),

$$\sin(a_2) = 1 - \cos(a_2)^2/2 - \cos(a_2)^4/8 - \cos(a_2)^6/16 \ldots \tag{3.145}$$

A good approximation for $\sin(a_2)$ can be obtained by truncating the series (3.145) at an appropriate length, substituting in the

expression for $\cos(a_2)$, and then retaining all polynomial terms of a given maximum order or less, say 4^{th} order. This is rather difficult to do, however, as a large number of the terms of (3.145) have significant components of 4^{th} order or lower.

Over the workspace the cosine of a_1 is,

$$\cos(a_1) = 1/2 - z_2/4 + z_1\sin(a_2)/(z_1^2/2+z_2^2/2-2z_2+2) \qquad (3.146)$$

The troublesome polynomial in the denominator of (3.146) can be replaced with its Taylor series equivalent taken about the origin of the workspace,

$$(z_1^2/2+z_2^2/2-2z_2+2)^{-1} = 1/2 + z_2/2 - z_1^2/8 + 3z_2^2/8 - 3z_1^2z_2/12 + 3z_2^3/12 \ldots \qquad (3.147)$$

The expression for $\cos(a_1)$ can then be truncated to those terms of 4^{th} order or lower. Given a sum of polynomials representation of $\cos(a_1)$, one can then obtain the sine of a_1 using a binomial expansion of the right hand side of (3.135), as was done for $\sin(a_1)$,

$$\sin(a_2) = 1 - \cos(a_2)^2/2 - \cos(a_2)^4/8 - \cos(a_2)^6/16 \ldots \qquad (3.148)$$

The factor $1/\sin(a_2)$ that appears in the inverse kinematics functions for $\dot{a}_1$, $\dot{a}_2$, $\ddot{a}_1$ and $\ddot{a}_2$ can be replaced with the Taylor series for the reciprocal of the polynomial approximation for $\sin(a_2)$, taken about the origin of the workspace.

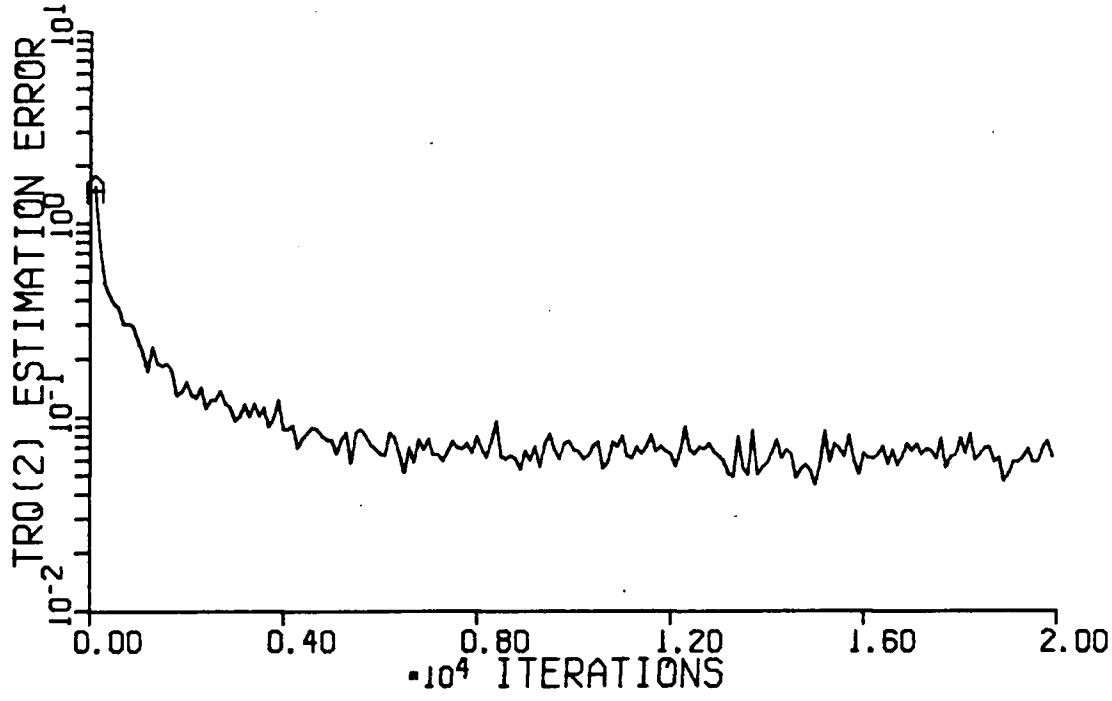
Accurate representation of these sine, cosine and cosecant terms can be obtained using the polynomial series shown here, truncated to a certain maximum order, 4^{th} order for example. Substituting these results into the inverse kinematics

relationships and subsequently combining the inverse kinematics with inverse dynamics, one can obtain an accurate sum of polynomials expression for the Cartesian inverse dynamics. Again, only terms of a certain maximum order or less need be retained. It is clearly difficult to perform the algebra necessary to derive such a sum of polynomials representation of the Cartesian inverse dynamics. The feasibility of such a representation can be demonstrated, however, by pre-learning a sum of polynomials representation of the Cartesian inverse dynamics.

3.6.5 PRE-LEARNING OF THE CARTESIAN INVERSE DYNAMICS

To demonstrate the adequacy of sum of polynomials representation of the Cartesian inverse dynamics, such a representation was pre-learned using the analytical Cartesian inverse dynamics as a guide. This was done using a program analagous to that used in chapter 2 to investigate the convergence rates of the various learning algorithms except that the target functions used were the two functions that make up the Cartesian inverse dynamics. Pre-learning was carried out using Method 2, Learning Identification, for the cases $\overset{*}{f} = \tau_1$ and $\overset{*}{f} = \tau_2$. The number of input variables was $v = 6$ and the system order was $s = 4$. The input variables were those given in equations (3.138) through (3.143). Training points were randomly generated uniformly over the space, $-1 < z_i < 1$, and 20,000 iterations were performed.

Figure 3.23 shows the reduction of Δf_1 and Δf_2 as a function of the number of training iterations performed. The



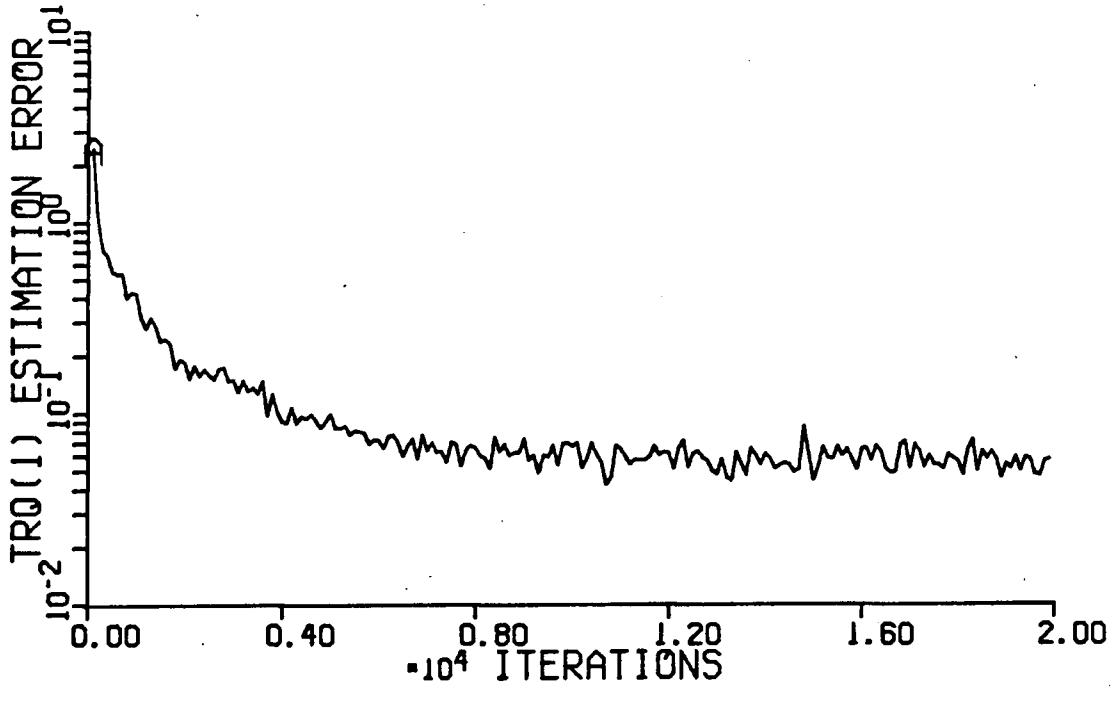


Figure 3.23 Reduction of estimation error during pre-learning of the Cartesian inverse dynamics (estimation error averaged over each 100 iteration interval)

graphs show the magnitudes of Δf_1 and Δf_2, averaged over intervals of 100 training iterations. The average error in estimating the target functions (in this case the torques τ_1 and τ_2) has dropped to about 0.6 N.m. It was found that the Cartesian inverse dynamics, as pre-learned here, were adequate to achieve control comparable to that using the analytical Cartesian inverse dynamics.

3.6.6 CLOSED LOOP CONTROL USING THE PRE-LEARNED CARTESIAN INVERSE DYNAMICS

To test the adequacy of the pre-learned Cartesian inverse dynamics, simulations were carried out in which the pre-learned Cartesian inverse dynamics were substituted for the analytical inverse in otherwise standard analytical control. Simulation parameters were set as follows: CLOSED=.TRUE., INVARM=.FALSE., EXACT=.FALSE., VISION=.TRUE., DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.FALSE., k_1=16, k_2=64 and t_{calc}=0.01 sec. The pre-learned coefficients were used in Learning Machine 1 to estimate torques. No learning took place during the simulations. Simulations were carried out for all six standard paths. The resulting path errors are shown in table 3.9. Mean position tracking error is about 3 mm compared to about 0.5 mm when using standard analytical control. Velocity and accelaration tracking errors are similarly increased. Also shown in table 3.9 is the error in estimating the torque and the maximum torque that is applied during each path. The error in the estimating the torque is typically about 4 percent of the maximum applied torque. Figure 3.24 shows a plot of path 1. It is evident that

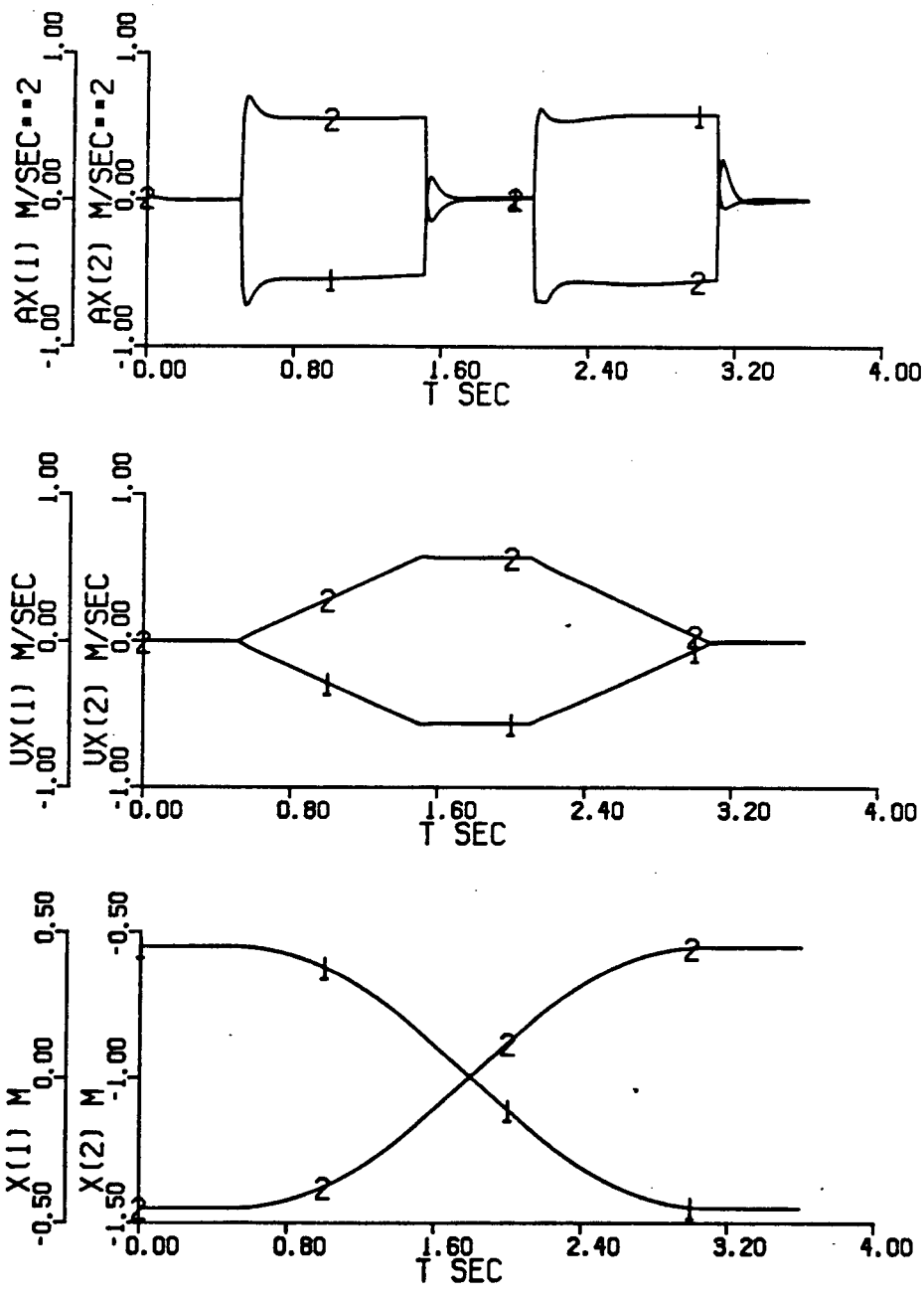


Figure 3.24 Closed loop control of line 1 using the pre-learned Cartesian inverse dynamics

control is quite stable as no large glitches are occuring other than some overshooting of steps in acceleration. While the performance is not quite as good as with standard analytical control, it is good enough to demonstrate that a sum of polynomials representation of the Cartesian inverse dynamics is adequate to achieve good control.

PATH	XERMS	VXERMS	AXERMS	TERMS	TRQMAX
line 1	3.82E-3	7.26E-3	9.94E-2	3.44E-1	8.33
line 2	8.88E-3	2.02E-3	1.36E-1	4.08E-1	9.15
line 3	6.05E-4	2.76E-3	1.01E-1	1.65E-1	14.1
line 4	3.32E-3	6.49E-3	1.01E-1	4.56E-1	6.79
circle 1	2.46E-3	6.19E-3	5.30E-2	3.21E-1	9.62
circle 2	2.01E-3	6.41E-3	5.56E-2	2.68E-1	13.6

Table 3.9 Path error and torque error using the pre-learned Cartesian inverse dynamics in closed loop control

3.7 ADEQUACY OF A SUM OF POLYNOMIALS REPRESENTATION OF THE DIRECT POSITION KINEMATICS

Another principal goal of this work is to learn a sum of polynomials representation of the direct kinematics of the two link manipulator without recourse to analysis of the manipulator. As with the inverse dynamics and inverse kinematics, it is instructive to first consider whether the

direct position kinematics can be adequately represented by a sum of polynomials for the purpose of replacing the vision system.

3.7.1 DERIVATION OF A SUM OF POLYNOMIALS REPRESENTATION OF THE DIRECT POSITION KINEMATICS

A sum of polynomials representation of the direct position kinematics can be derived quite easily. Since the direct kinematics of any manipulator can be obtained in a similar form by a general technique, it would appear that a sum of polynomials representation of the direct kinematics is achievable for all manipulators.

The analytical direct position kinematics are given by equations (3.32) and (3.33). For a two link manipulator with parameters as given in equations (3.99) through (3.103), the analytical direct kinematics are as follows,

$$x_1 = \sin(a_1) + \sin(a_1 + a_2) \tag{3.149}$$

$$x_2 = -\cos(a_1) - \cos(a_1 + a_2) \tag{3.150}$$

Since we are concerned with modelling the direct position kinematics over the workspace in particular, the input variables a sum of polynomials representation can be chosen as follows,

$$z_1 = (a_1 + \pi/3)/(\pi/2) \tag{3.151}$$

$$z_2 = (a_2 - 2\pi/3)/(\pi/2) \tag{3.152}$$

$\langle z_1, z_2 \rangle = \langle 0,0 \rangle$ corresponds to the center of the workspace and the workspace is enclosed in the region defined by $-1 < z_i < 1$. With such a normalization of the input variables the direct position kinematics become,

$$x_1 = [\sin(\pi z_1/2) - \sqrt{3}\cos(\pi z_1/2) + \sin(\pi z_1/2)\cos(\pi z_2/2) + \cos(\pi z_1/2)\sin(\pi z_2/2) + \sqrt{3}\cos(\pi z_1/2)\cos(\pi z_2/2) - \sqrt{3}\sin(\pi z_1/2)\sin(\pi z_2/2)]/2 \quad (3.153)$$

$$x_2 = [-\cos(\pi z_1/2) - \sqrt{3}\sin(\pi z_1/2) - \cos(\pi z_1/2)\cos(\pi z_2/2) + \sin(\pi z_1/2)\sin(\pi z_2/2) + \sqrt{3}\sin(\pi z_1/2)\cos(\pi z_2/2) + \sqrt{3}\cos(\pi z_1/2)\sin(\pi z_2/2)]/2 \quad (3.154)$$

The sine and cosine terms can be replaced with their series equivalents, truncated to 4^{th} order or lower terms,

$$\sin(\pi z_1/2) \simeq \pi z_1/2 - \pi^3 z_1^3/48 \quad (3.155)$$

$$\cos(\pi z_1/2) \simeq 1 - \pi^2 z_1^2/4 + \pi^4 z_1^4/384 \quad (3.156)$$

$$\sin(\pi z_2/2) \simeq \pi z_1/2 - \pi^3 z_1^3/48 \quad (3.157)$$

$$\cos(\pi z_2/2) \simeq 1 - \pi^2 z_1^2/4 + \pi^4 z_1^4/384 \quad (3.158)$$

Substituting these expressions into equations (3.153) and (3.154), and keeping those terms of 4^{th} order or lower yields a sum of polynomials representation of the direct position kinematics,

$$x_1 = \pi z_1/2 + \pi z_2/4 - \sqrt{3}\pi^2 z_1 z_2/8 - \sqrt{3}\pi^2 z_2^2/16 - \pi^3 z_1^3/48 - \pi^3 z_2^3/96 - \pi^3 z_1^2 z_2/32 - \pi^3 z_1 z_2^2/32 + \sqrt{3}\pi^4 z_1^3 z_2/192 + \sqrt{3}\pi^4 z_1 z_2^3/192 + \sqrt{3}\pi^4 z_1^2 z_2^2/128 + \sqrt{3}\pi^4 z_2^4/768 \quad (3.159)$$

$$x_2 = -1 + \sqrt{3}\pi z_2/4 + \pi^2 z_1^2/8 + \pi^2 z_2^2/16 - \sqrt{3}\pi^3 z_1^2 z_2/32 - \sqrt{3}\pi^3 z_1 z_2^2/64 + \pi^2 z_1 z_2/8 - \sqrt{3}\pi^3 z_2^3/96 - \pi^4 z_1^4/384 - \pi^4 z_1^3 z_2/192 - \pi^4 z_1^2 z_2^2/128 - \pi^4 z_1 z_2^3/192 - \pi^4 z_2^4/768 \quad (3.160)$$

To test the accuracy of the derived direct position kinematics, simulations were carried out using standard analytical control for the six standard paths while using the

derived direct position kinematics to estimate the position at successive points in time. As with the modelled vision system, velocity and acceleration estimates are obtained by simple differentiating of the position estimates. Table 3.10 shows the error in path estimation for the various standard paths. Estimates of velocity and acceleration compare well with those obtained using the modelled vision system in which position estimates are exact. Errors in path estimation by the vision system are shown in table 3.11. Figure 3.25 shows the resulting view of standard path circle 1 using the derived direct position kinematics.

PATH	LXERMS	LVERMS	LAERMS
line 1	4.89E-3	7.75E-3	5.25E-2
line 2	3.43E-2	3.64E-2	1.09E-1
line 3	8.69E-4	3.42E-3	4.82E-2
line 4	1.28E-2	1.65E-2	6.92E-2
circle 1	6.60E-3	1.53E-3	6.08E-2
circle 2	6.58E-3	1.43E-3	5.70E-2

Table 3.10 Path estimation error using the derived direct position kinematics

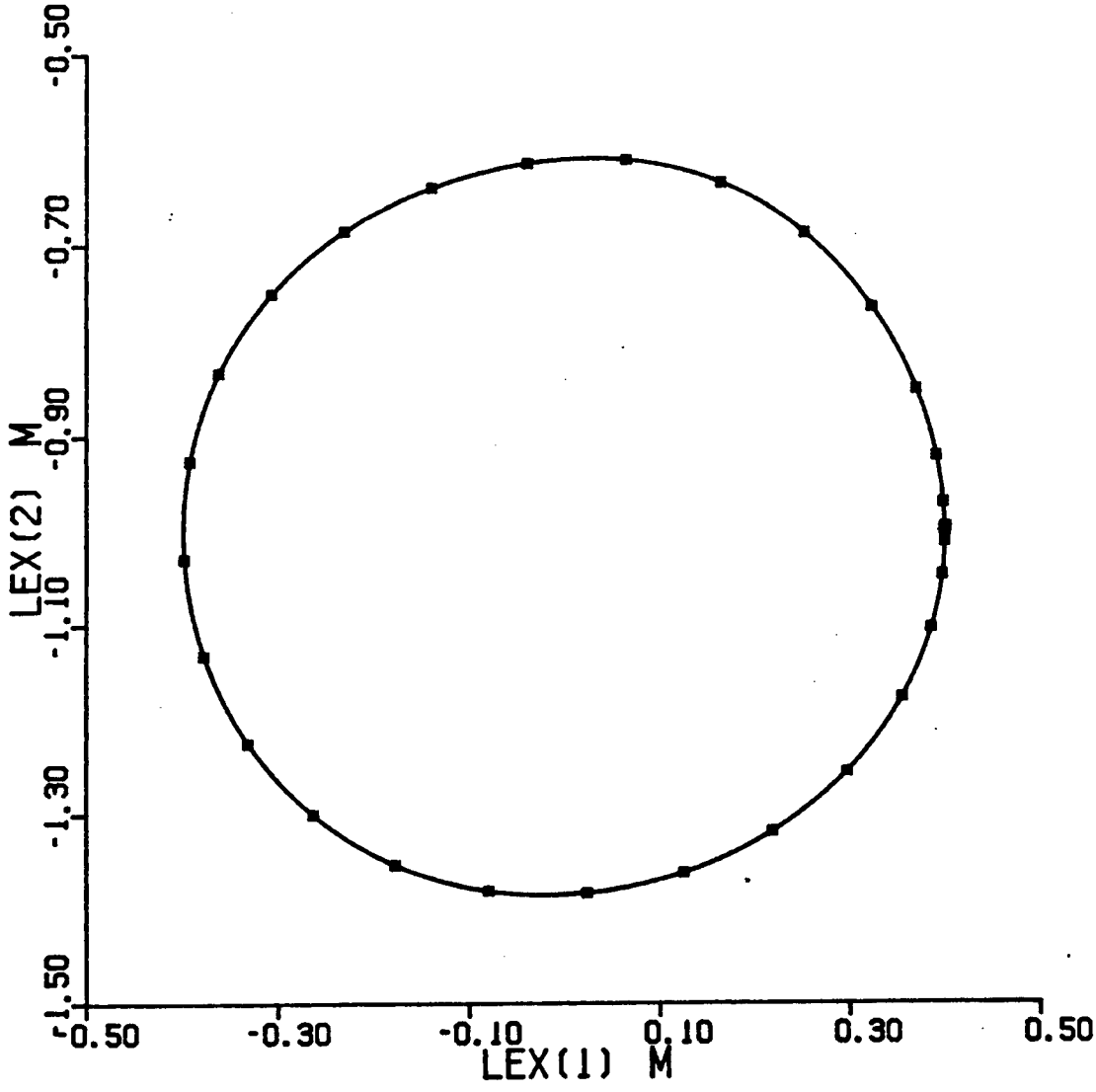


Figure 3.25 View of circle 1 using the derived direct position kinematics

PATH	LXERMS	LVERMS	LAERMS
line 1	0.0	3.01E-3	4.62E-2
line 2	0.0	3.02E-3	4.64E-2
line 3	0.0	3.21E-3	4.80E-2
line 4	0.0	3.20E-3	4.56E-2
circle 1	0.0	2.90E-3	2.56E-2
circle 2	0.0	2.89E-3	2.57E-2

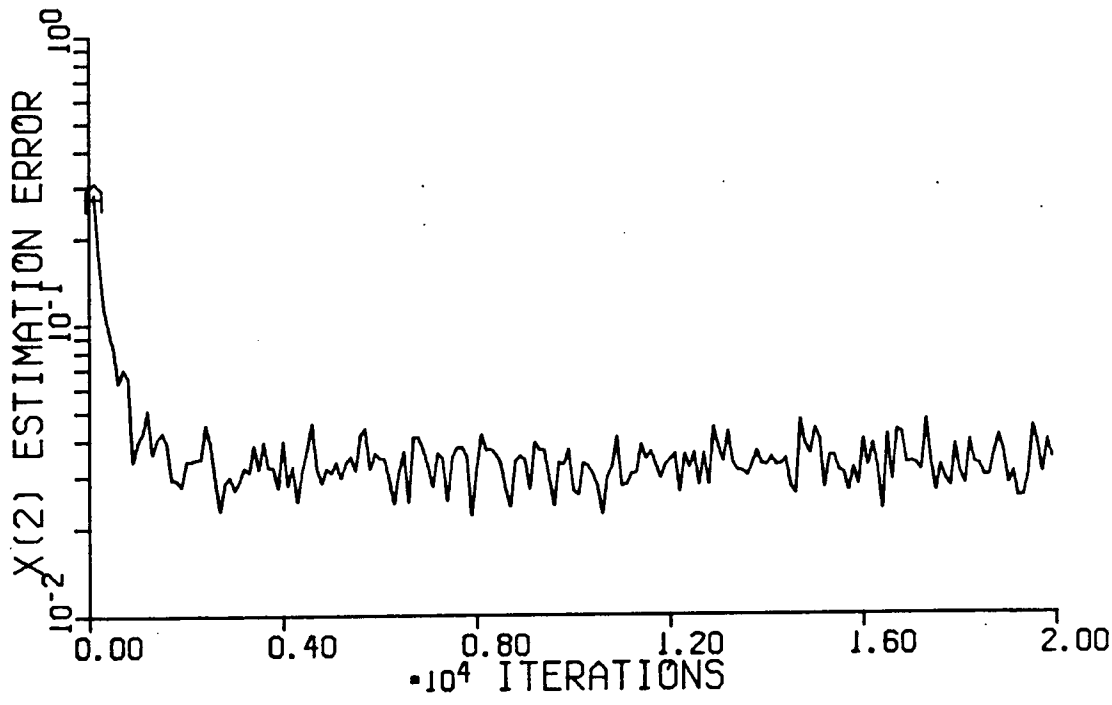
Table 3.11 Path estimation error using the vision system which has exact position estimation

3.7.2 PRE-LEARNING OF THE DIRECT POSITION KINEMATICS

Having derived a sum of polynomials representation of the direct position kinematics, it was verified that this representation could be pre-learned using the analytical direct position kinematics as a guide. This was done using a program analagous to that used in chapter 2 to investigate the convergence rates of the various learning algorithms except that the target functions used were the two functions that make up the direct position kinematics. Method 2, Learning Identification, was used for the cases, $\overset{*}{f} = x_1$ and $\overset{*}{f} = x_2$. The number of input variables was $v = 6$ and the system order was $s = 4$. The input variables were those given in equations (3.151) and (3.152). Training points were randomly generated uniformly over the space, $-1 < z_i < 1$, and 30,000 iterations were performed.

Figure 3.26 shows the reduction of the estimation errors, Δf_1 and Δf_2, as a function of the number of iterations. The graphs show the magnitudes of Δf_1 and Δf_2, averaged over intervals of 100 training iterations. The average error in estimating the target functions (in this case the positions x_1 and x_2) decreases to about 0.03 m. This represents the best accuracy achievable with a 4^{th} order system as further iterations were not reducing the error.

Table 3.12 lists the pre-learned coefficients alongside the derived coefficients from equations (3.159) and (3.160). It can be seen that the learned coefficients are quite close to those derived previously. Given the similarity in coefficients, one would expect that path estimation errors using the pre-learned direct position kinematics would be similar to those that occurred previously with the derived direct position kinematics. Table 3.13 shows the errors in path estimation for the various standard paths when simulated using standard analytical control and viewed with the pre-learned direct position kinematics. Figure 3.27 shows the resulting view of circle 1.



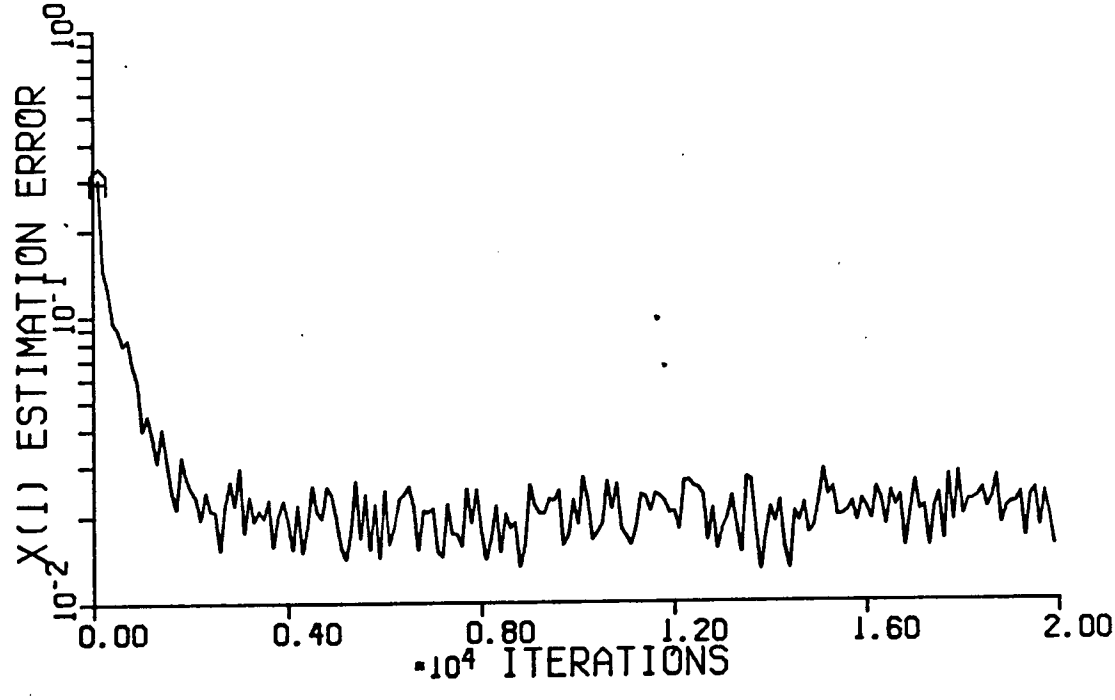


Figure 3.26 Reduction of estimation error during pre-learning of the direct position kinematics (estimation error averaged over each 100 iteration interval)

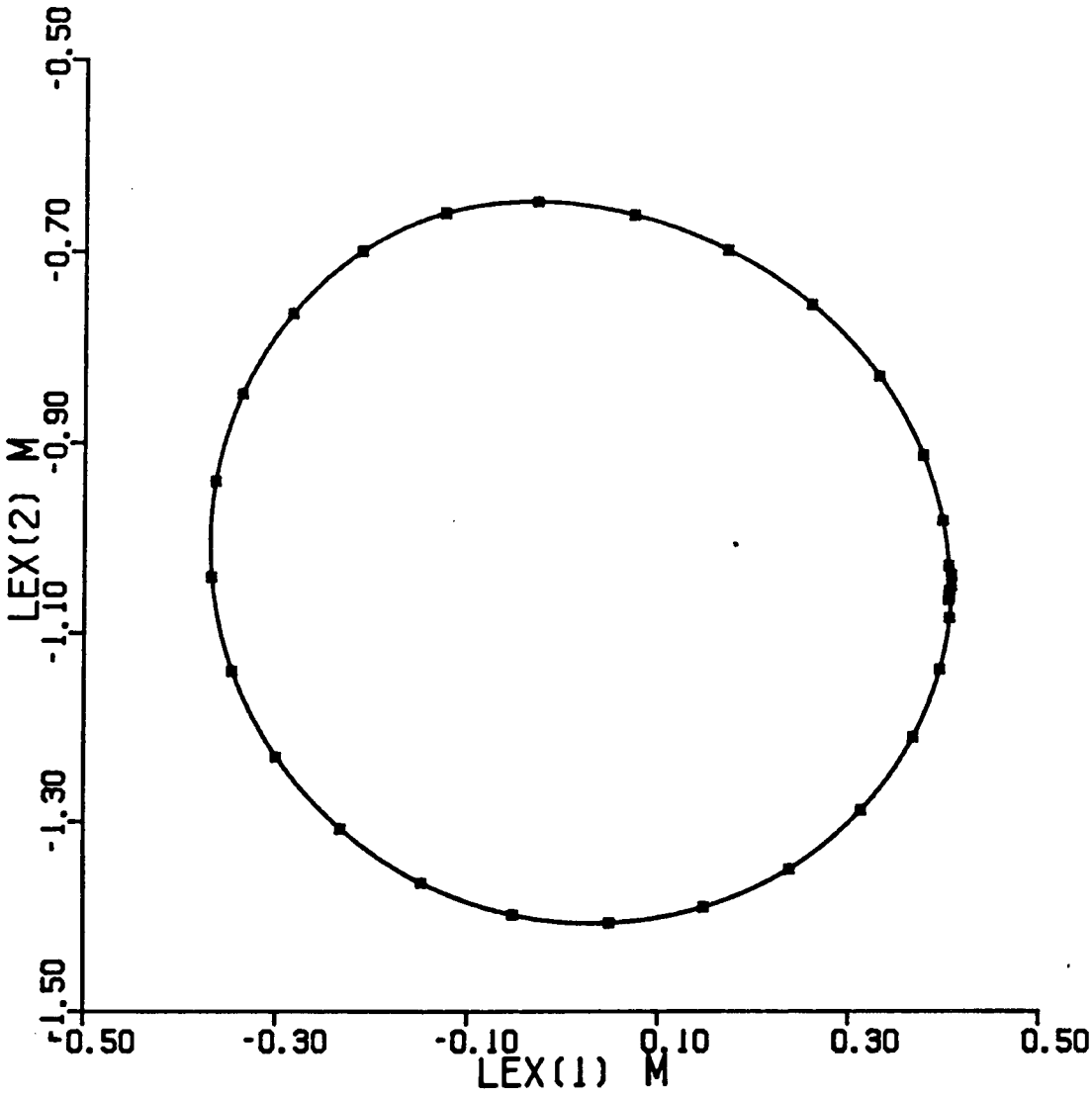


Figure 3.27 View of circle 1 using the pre-learned direct position kinematics

POLYNOMIAL TERM	X(1) COEFFICIENTS DERIVED	X(1) COEFFICIENTS PRE-LEARNED	X(2) COEFFICIENTS DERIVED	X(2) COEFFICIENTS PRE-LEARNED
1	0.0	0.020	-1.00	-1.02
z_1	1.57	1.50	0.0	-0.082
z_1^2	0.0	0.044	1.23	1.18
z_1^3	-0.646	-0.496	0.0	0.133
z_1^4	0.0	0.056	-0.254	-0.226
z_2	0.785	0.730	1.36	1.26
z_2z_1	-2.14	-2.04	1.23	1.14
$z_2z_1^2$	-0.969	-0.668	-1.68	-1.12
$z_2z_1^3$	0.879	0.613	-0.507	-0.398
z_2^2	-1.07	-1.02	0.617	0.580
$z_2^2z_1$	-0.969	-0.667	-0.839	-1.16
$z_2^2z_1^2$	1.32	0.984	-0.761	-0.542
z_2^3	-0.323	-0.221	-0.559	-0.364
$z_2^3z_1$	0.879	0.632	-0.507	-0.380
z_2^4	0.220	0.146	-0.127	-0.112

Table 3.12 Coefficients for derived and pre-learned sum of polynomials representations of the direct position kinematics functions for positions x_1 and x_2.

PATH	LXERMS	LVERMS	LAERMS
line 1	3.17E-2	1.59E-2	5.78E-2
line 2	4.49E-2	3.11E-2	7.67E-2
line 3	3.74E-2	2.52E-2	7.00E-2
line 4	3.27E-2	1.79E-2	5.37E-2
circle 1	3.74E-2	2.18E-2	4.85E-2
circle 2	3.73E-2	2.40E-2	4.98E-2

Table 3.13 Path estimation error using the pre-learned direct position kinematics

3.7.3 CLOSED LOOP CONTROL USING THE PRE-LEARNED CARTESIAN INVERSE DYNAMICS AND PRE-LEARNED DIRECT POSITION KINEMATICS

Simulations verified that closed loop control is possible using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics. Simulation parameters were set as follows: CLOSED=.TRUE., INVARM=.FALSE., EXACT=.FALSE., VISION=.FALSE., DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.FALSE., k_1=16, k_2=64 and t_{calc}=0.01 sec. All six standard paths were simulated. The resulting path errors are shown in table 3.14.

PATH	XERMS	VXERMS	AXERMS
line 1	2.77E-2	2.27E-2	1.78E-1
line 2	4.08E-2	6.13E-2	3.10E-1
line 3	3.66E-2	5.87E-2	3.16E-1
line 4	2.82E-2	2.83E-2	1.87E-1
circle 1	3.52E-2	3.24E-2	1.56E-1
circle 2	3.37E-2	3.19E-2	1.41E-1

Table 3.14 Path error using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics in closed loop control

Tracking is significantly worse than that achieved with the vision system, however, this is to be expected due to the inaccuracies of the pre-learned direct position kinematics. Path position errors are of the same order of magnitude as position estimation errors, as shown previously in table 3.12. The control system perceives that it is forcing the manipulator to closely follow the path specification. For example, figure 3.28 shows the resulting path as seen using the pre-learned direct position kinematics. Standard path circle 1 appears to be followed closely. Due to the inaccuracies in the pre-learned direct position kinematics, the actual path is somewhat different, as shown in figure 3.29. The manipulator must move through a distorted trajectory such that the perceived trajectory follows the path specification.

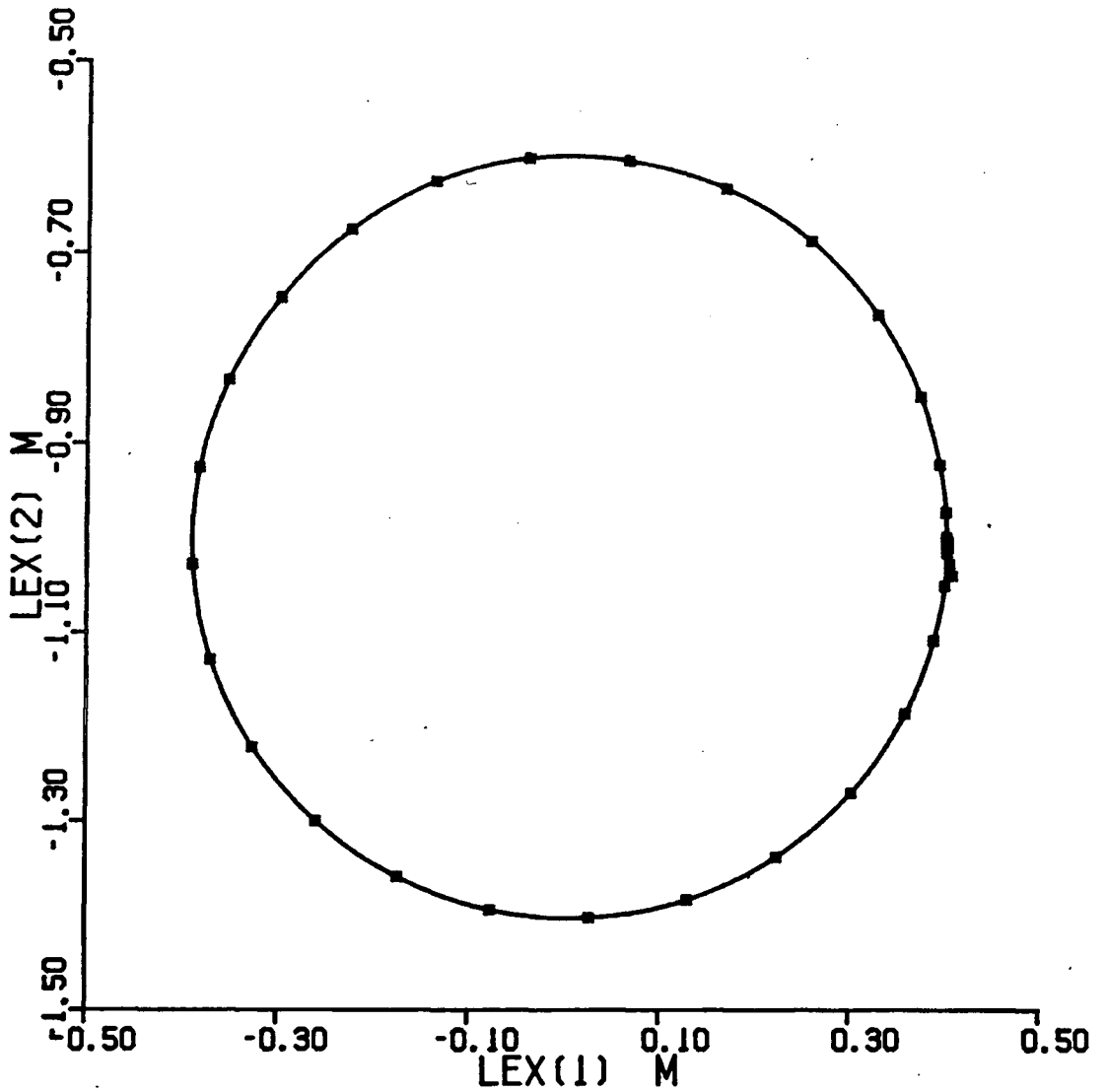


Figure 3.28 View of circle 1 using the pre-learned direct position kinematics during closed loop control of circle 1 using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics

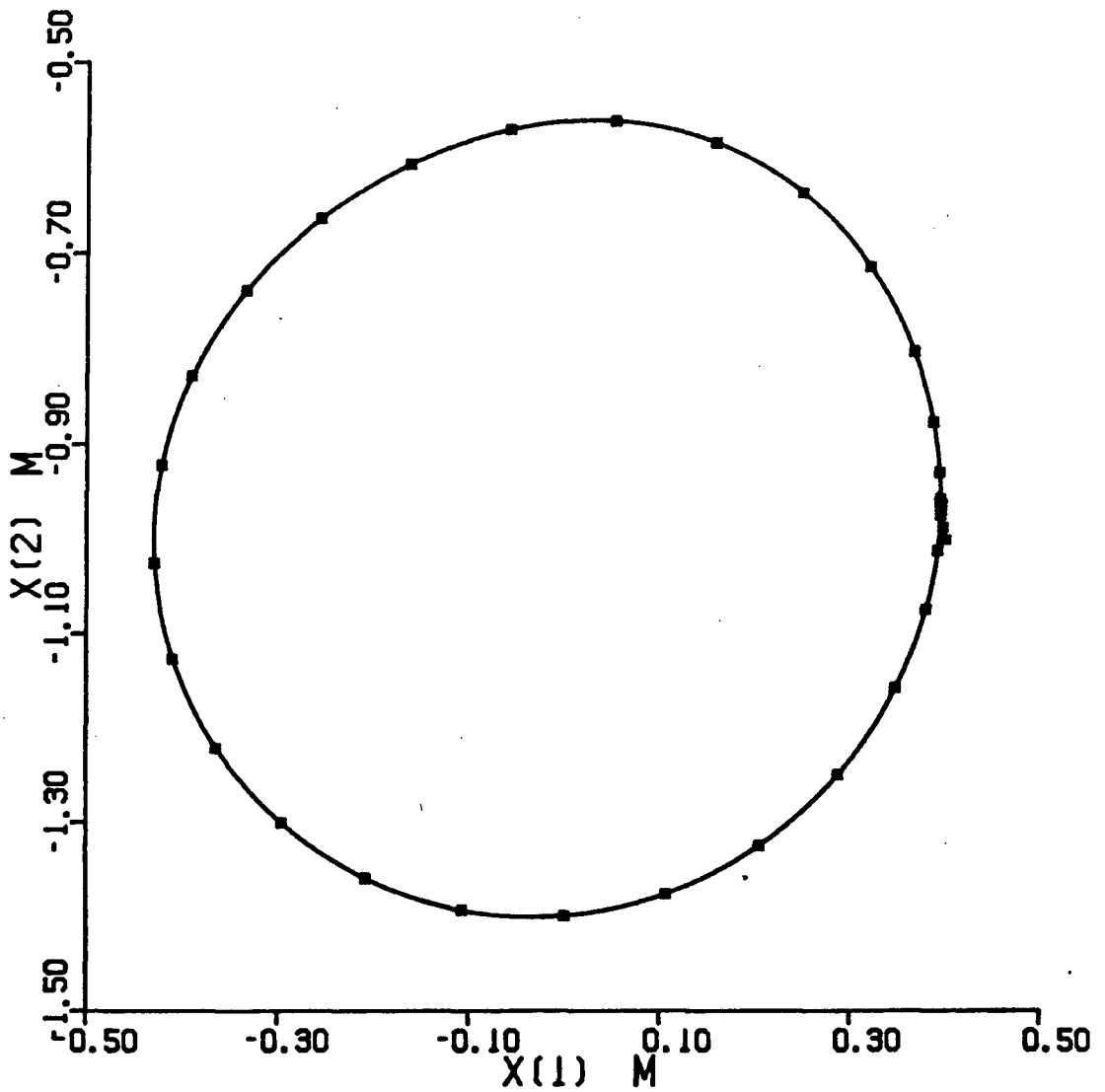


Figure 3.29 View of circle 1 during closed loop control of circle 1 using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics

Figure 3.30 shows the resulting path for the simulation of line 1. The large accelerations that initially occur are due to the sudden perceived position error that results when one switches from the accurate vision system to the less accurate pre-learned direct position kinematics. This undesirable glitch can be eliminated by using Interference Minimization or Learning Identification to perform one iteration of additional learning of the direct position kinematics before switching to use of the pre-learned direct position kinematics for feedback purposes. This is invoked by setting logical variable ADJVIS to be true. Since Interference Minimization and Learning Identification eliminate estimation error at the training point, this avoids the sudden perceived position error when subsequently switching to the pre-learned direct position kinematics. Figure 3.31 shows a simulation of line 1 using this technique. Notice how tracking is improved near the training point but minimally affected away from the training point. Similar glitches at the end of a path where the pre-learned direct position kinematics are used can be avoided by measuring the final position error with the vision system and then executing a short corrective path using the vision system for feedback purposes.

In figures 3.30 and 3.31 there are several other unexplained glitches in the acceleration profiles. These occur because the learned Cartesian inverse that drives the manipulator is doing only a nominal job of cancelling the direct dynamics and the direct kinematics based on the learned direct position kinematics. It appears that at certain points in path

space, the mismatch in the inverse and direct components is causing feedback to become positive, thus briefly reinforcing errors until such problematic points in path space are left. Clearly, if the pre-learned direct position kinematics were much less accurate, control would become unstable over large regions of path space and path specifications could not be followed. Nevertheless, the simulations show that coarse control is possible using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics. Finer control appears to require only an improvement in the accuracy of the learned direct position kinematics.

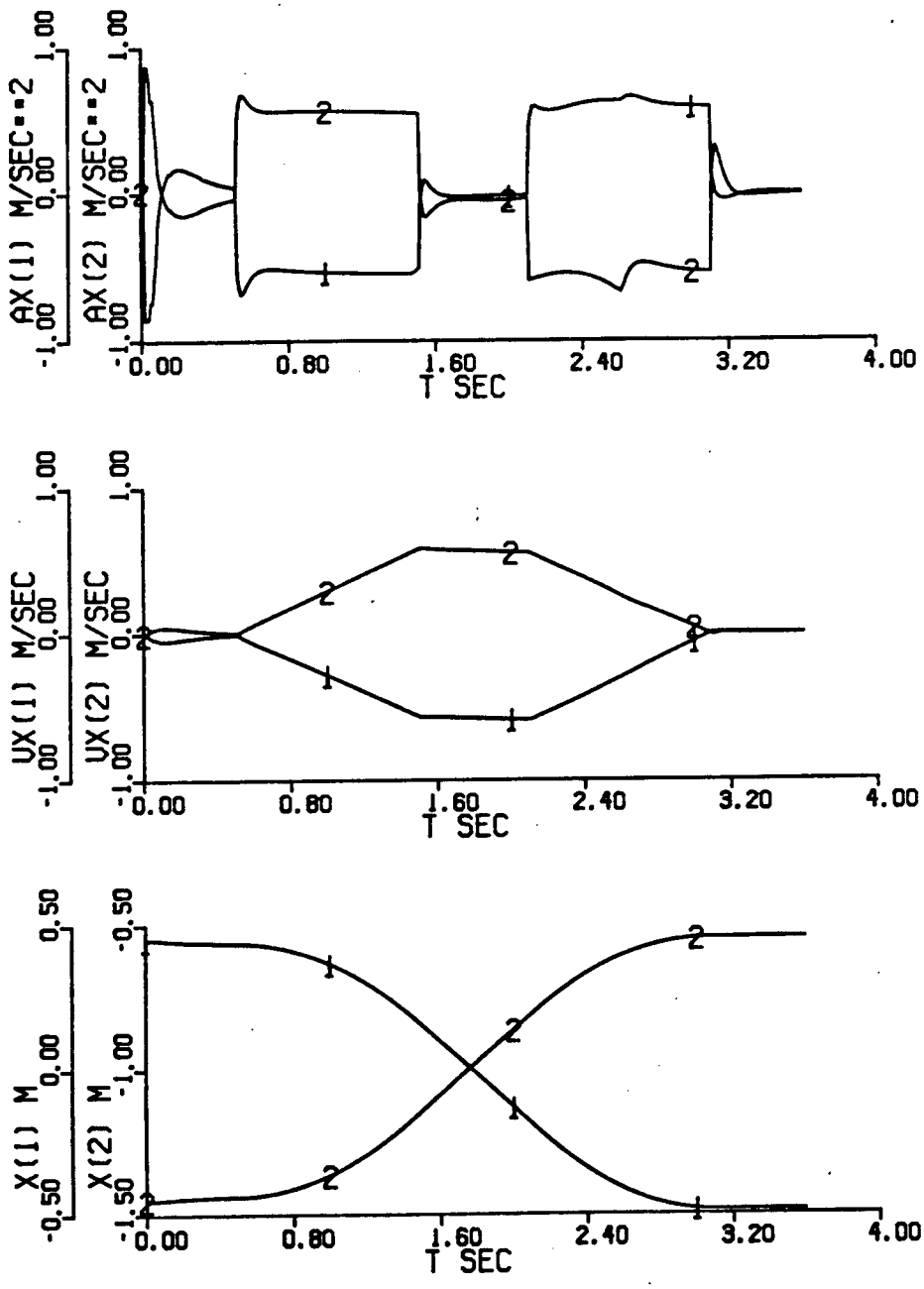


Figure 3.30 Closed loop control of line 1 using the pre-learned Cartesian inverse dynamics and pre-learned direct position kinematics

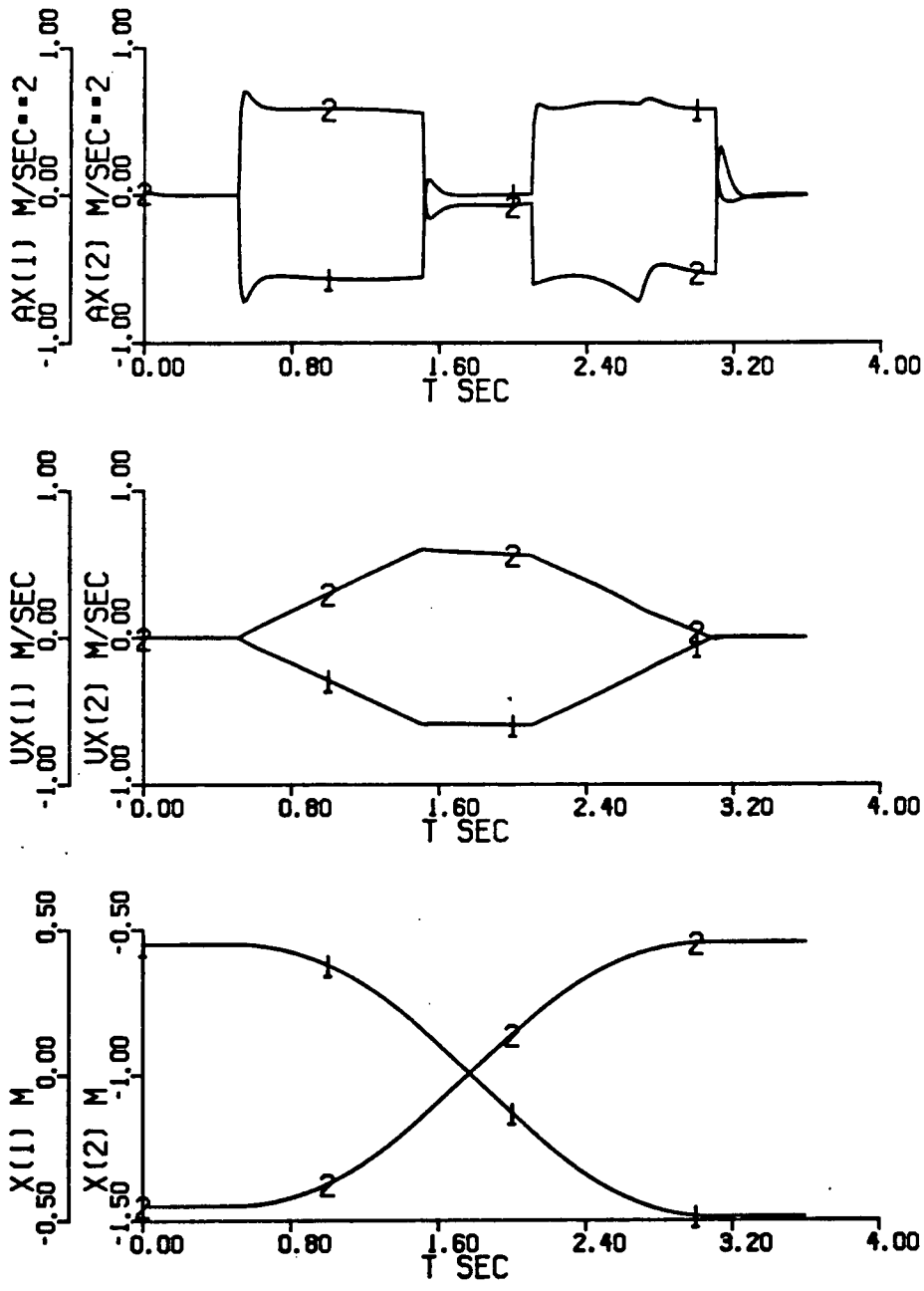


Figure 3.31 Closed loop control of line 1 using the pre-learned Cartesian inverse dyanmics and pre-learned direct position kinematics with ADJVIS=.TRUE.

3.8 SELF-LEARNING OF THE CARTESIAN INVERSE DYNAMICS

Having demonstrated that a sum of polynomials representation of the Cartesian inverse dynamics is adequate to achieve good control, it remains only to find a method of learning the Cartesian inverse dynamics without recourse to analysis of the manipulator. We have found that this can be done through observation of applied torques and corresponding manipulator motion; hence the name self-learning.

3.8.1 A METHOD FOR SELF-LEARNING OF THE CARTESIAN INVERSE DYNAMICS

THE METHOD

Given sufficient and varied observations of the manipulator position, velocity, torques and resulting acceleration, one can use Interference Minimization or related methods to learn the Cartesian inverse dynamics. The correspondence between position, velocity, acceleration and torque imposed by the direct dynamics and direct kinematics, ie. the manipulator and the vision system, is the same as that imposed by the Cartesian inverse dynamics; an n-tuple of observations valid for the direct relationships is also a valid n-tuple for the inverse relationships.

To learn the Cartesian inverse dynamics over a path space such as that defined by equations (3.138) through (3.143), two conditions must be met: First, one must obtain observations corresponding to all regions of the path space, $-1 < z_i < 1$. Here the z_i are the normalized Cartesian positions, velocities and accelerations that act as input variables in the sum of

polynomials representation of the Cartesian inverse dynamics. Observations corresponding to the infinite number of points in path space are fortunately not necessary; the generalization that occurs when using Interference Minimization or related methods will effectively interpolate between training points. There must be sufficient and varied observations, however, to allow accurate interpolation over the whole of path space and to overcome the learning interference that occurs during training. Secondly, one must constrain manipulator motion to be within the bounds $-1 < z_i < 1$ in order that learning can take place. Such constraints cannot be imposed using the as yet unlearned Cartesian inverse dynamics and must not require use of the analytical Cartesian inverse dynamics.

A simple method has been found for restoring manipulator motion to be within the path space $-1 < z_i < 1$ whenever it is observed that manipulator motion has exceeded the bounds of this path space. The method meets the above mentioned requirements.

The center of path space is defined by z_i equal to zero. This corresponds to zero velocity and zero acceleration while at Cartesian position,

$$\langle x_{c1}, x_{c2} \rangle = \langle 0, -1 \rangle \text{ m} \tag{3.161}$$

This corresponds to joint position,

$$\langle a_{c1}, a_{c2} \rangle = \langle 2\pi/3, -\pi/3 \rangle \tag{3.162}$$

By simple static analysis of the manipulator, the corresponding torques are,

$$\tau_{c1} = -8.50 \text{ N}\cdot\text{m} \tag{3.163}$$

$$\tau_{c2} = 8.50 \text{ N}\cdot\text{m} \tag{3.164}$$

One can apply uncoupled joint servo control to maintain the manipulator at joint position $\langle a_{c1}, a_{c2} \rangle$ or restore the manipulator to this position if disturbed from it. An example of such uncoupled joint servo control is the following,

$$\tau_1 = \tau_{c1} - j1\dot{a}_1 - j2(a_1 - a_{c1}) \qquad (3.165)$$

$$\tau_2 = \tau_{c2} - j1\dot{a}_2 - j2(a_2 - a_{c2}) \qquad (3.166)$$

Since the servoed joint position corresponds to the center of the Cartesian path space, such simple joint servo control can be used to return the manipulator within the bounds $-1 < z_i < 1$. The angular position is easily measured with potentiometers and the angular velocity can be obtained from these measurements by simple differentiation.

Given a method for constraining the manipulator within the path space or more correctly returning the manipulator within the path space when it is observed to be out of bounds, it is only necessary to find a mechanism to cause the manipulator to move so as to explore all the regions of path space. We have found that this can be done by learning the sum of polynomials representation of the Cartesian inverse dynamics while using the same partially learned Cartesian inverse dynamics to drive the manipulator over a sequence of randomly generated paths that encompass the regions of path space. Initially, when the self-learned Cartesian inverse dynamics are very poorly known, the path specifications are not followed and constraining action is frequently required to restore the manipulator within the bounds $-1 < z_i < 1$. During such poorly controlled thrashing, however, learning can take place whenever the manipulator is

within bounds. In fact the manipulator acceleration may immediately become out of bounds when driven by the as yet unlearned Cartesian inverse dynamics; learning may initially only be possible at those instances when the constraining action has just returned the manipulator within bounds. Later, when the self-learned Cartesian inverse dynamics become more accurate, the path specifications are followed more closely. The self-learned Cartesian inverse dynamics are eventually learned over the whole of path space if the training paths cover the whole of path space.

Simulations were performed of 1000 training paths during which self-learning of the Cartesian inverse dynamics was carried out. Simulation parameters were set as follows: CLOSED=.TRUE., INVARM=.FALSE., EXACT=.FALSE., VISION=.TRUE., DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.FALSE., k_1=4, k_2=4, and t_{calc}=0.01 sec. The feedback gains were reduced somewhat from those used in standard analytical control as this permitted stable control to be achieved earlier in the training process; higher feedback gains cause better path tracking when the Cartesian inverse dynamics have been well learned but create more instability during earlier stages of training when the Cartesian inverse dynamics are only partially learned.

Learning parameters were set as follows: LRNINV=.TRUE., t_{lrn1}=0.01 sec and Δf_{min}=0.01 N·m. Using the nomenclature of chapter 2, Method 2, Learning Identification, was the learning algorithm used. The number of input variables was v = 6 and the system order was s = 4. The input variables were those given in

equations (3.138) through (3.143).

Constraining action was invoked whenever the manipulator was observed to be out of bounds. The gains used in the uncoupled joint servo control defined in (3.165) and (3.166) were,

$$j_1 = 4 \tag{3.167}$$

$$j_2 = 4 \tag{3.168}$$

In addition the magnitudes of the applied torques were restricted at all times to be less than TRQLIM = 24 N·m. This models the finite drive capability present in real manipulators; in practice applied torques are limited by the motors, hydraulics, or pneumatics used.

At each interval of t_{lrn1}, learning was conditionally performed. First, the observed manipulator motion had to be within the bounds $-1 < z_i < 1$ in order that learning could take place. Then the coefficients of the partially learned sum of polynomials representation of the Cartesian inverse dynamics were used off-line to estimate the torques corresponding to the observed manipulator motion. Initially these coefficients were all zero. By off-line we mean that these estimated torques are not applied to the manipulator. Note that this means that self-learning does not have to be done in real time. In self-learning of the Cartesian inverse dynamics, the estimation error, Δf, is the difference between the estimated torque and the corresponding torque that was applied to the manipulator at the instant that the motion was observed. If this estimation error was greater than Δf_{min} for either of the torques then the sum of

polynomials representation of the Cartesian inverse dynamics function for the corresponding torque was adjusted using Learning Identification.

Note that manipulator motion was assumed to be measured exactly for the purposes of self-learning. This is not such an unreasonable assumption as it might seem. Because self-learning is done off-line without real time contraints, one can devote more computer time to obtaining accurate estimates of velocity and acceleration based on changes in observed position. Since self-learning from a point of no knowledge of the Cartesian inverse dynamics is a one time event, occurring before a manipulator is put to work, it is reasonable to assume that additional computing capability can temporarily be utilized. Note also that one can base acceleration and velocity estimates on observations of position, after, as well as before, the point in time in question. Thus it is reasonable to assume that observations of manipulator motion for self-learning purposes can be made much more accurately than observations for use in closed loop control.

Linear paths were used for training purposes. The acceleration specification for each path was chosen uniformly between the limits,

$$0.1 < a_{max} < 0.9 \text{ m/sec}^2 \qquad (3.169)$$

The velocity specifications were chosen uniformly between the limits,

$$0.05 < v_{max} < 0.95 \text{ m/sec} \qquad (3.170)$$

a_{max} and v_{max} could not be chosen too close to zero as the path

specification would take an inordinate amount of time to be carried out. The final position of each path was chosen uniformly from within the region defined by,

$$0.375 < \text{abs}(x_{fi1}-x_{c1}) < 0.45 \text{ m} \quad (3.171)$$

$$0.375 < \text{abs}(x_{fi2}-x_{c2}) < 0.45 \text{ m} \quad (3.172)$$

Note that the final position of each path is the initial position of each subsequent path. The upper limits on acceleration, velocity, and position serve to avoid path specifications that are near the bounds of path space and thus likely to invoke constraining action even when tracking errors are quite small. Figure 3.32 shows the randomness of the choices of a_{max} and v_{max} for the first 250 training paths. The restriction that paths begin and end in a band near the edge of the workspace results in a fairly uniform coverage of positions within the workspace by the training paths. Figure 3.33 shows the positions of the first 250 training paths.

RESULTS

During the initial stages of self-learning, the manipulator is frequently out of bounds and thus learning cannot take place at every interval of t_{lrn1}. Later, as better control is achieved, the proportion of opportunities at which learning takes place increases. Finally, as the self-learned Cartesian inverse dynamics become quite accurate, the proportion of opportunities at which learning takes place decreases since the estimation error is frequently less than the threshold, Δf_{min} = 0.01 N·m. The proportion of opportunities during each path at which learning takes place is shown in figure 3.34.

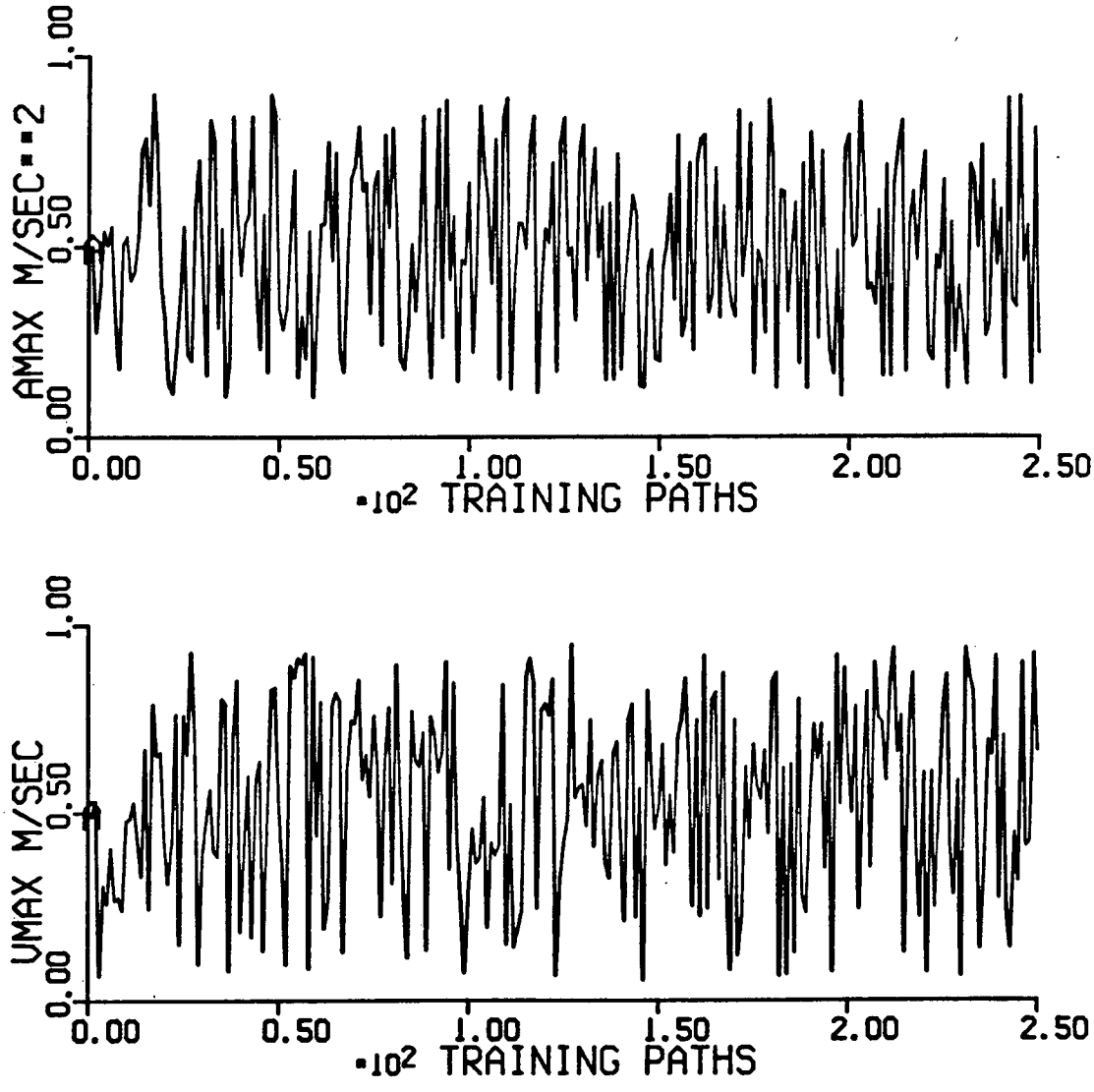


Figure 3.32 Choices of a_{max} and v_{max} for first 250 training paths used in self-learning of the Cartesian inverse dynamics

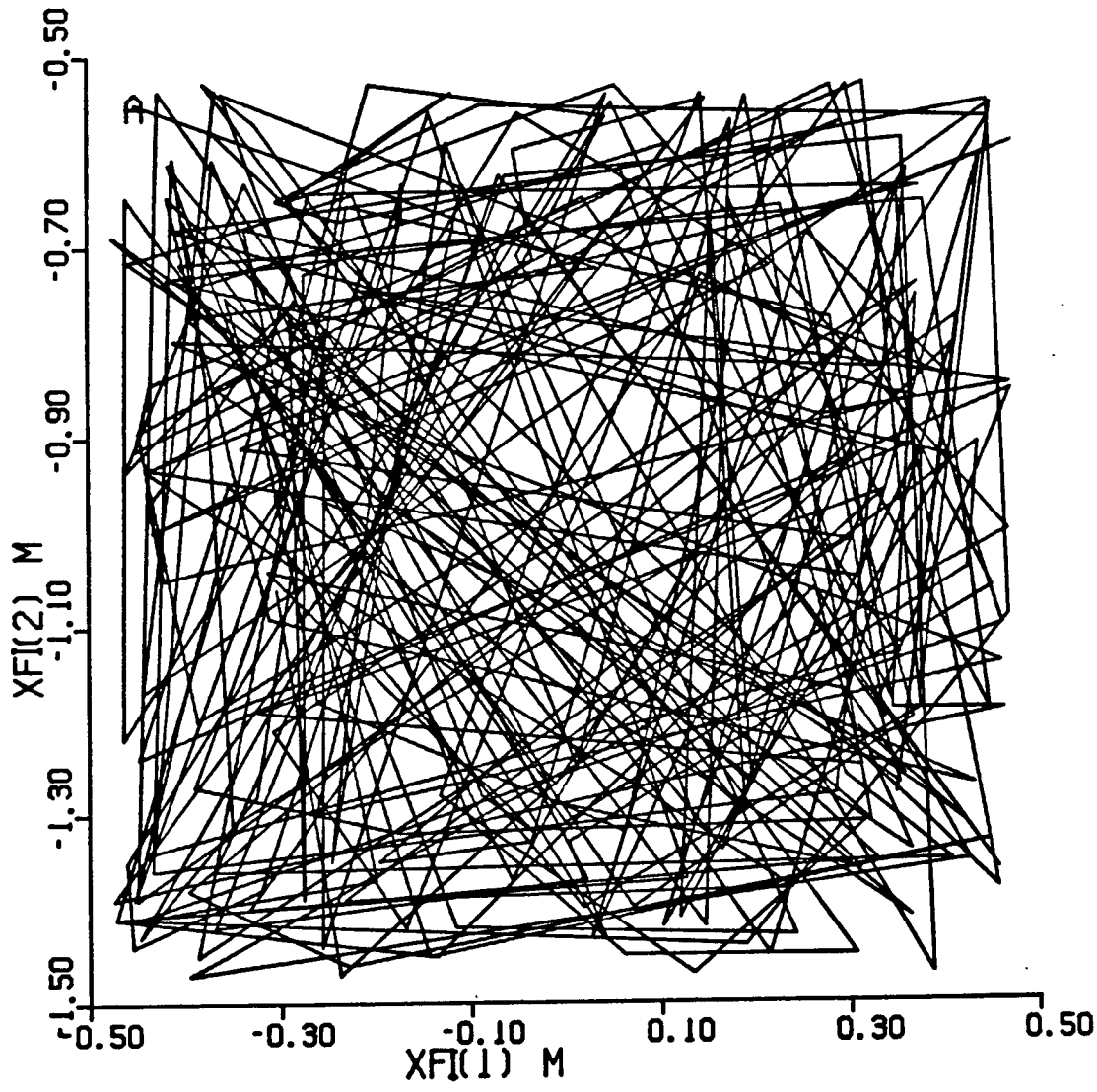


Figure 3.33 View of first 250 training paths used in self-learning of the Cartesian inverse dynamics

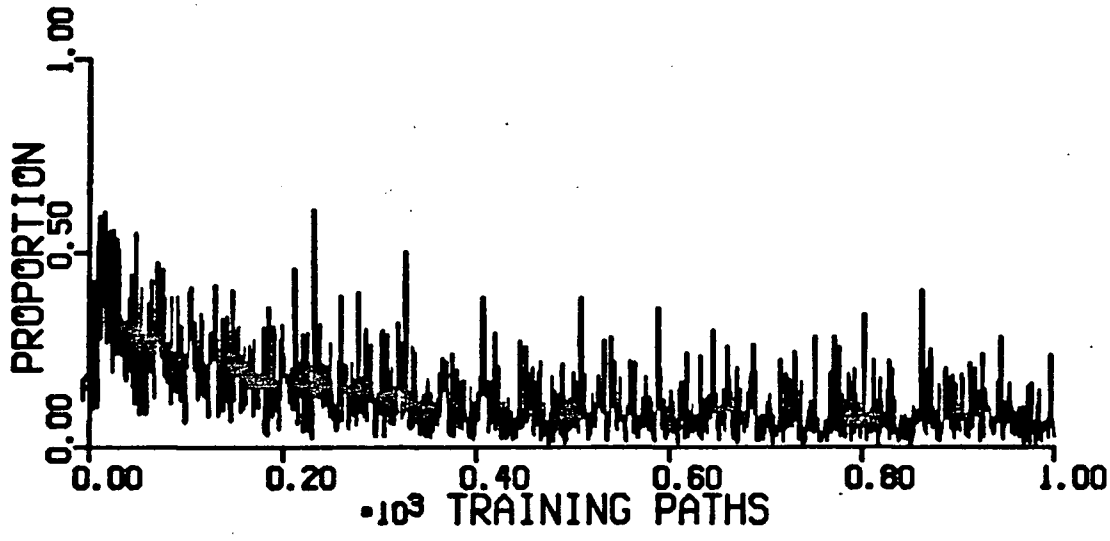


Figure 3.34 Proportion of learning opportunities at which learning took place during self-learning of the Cartesian inverse dynamics

As self-learning proceeds, the path specifications are followed more and more closely. Figure 3.35 shows the path errors that occurred during each of the training paths. Path errors are clearly being reduced as training takes place. Better control also means that the manipulator goes out of bounds less often. Figure 3.36 shows the maximum out of bounds excursions that occurred during each of the training paths. The frequency and extent of out of bounds excursions are reduced with training.

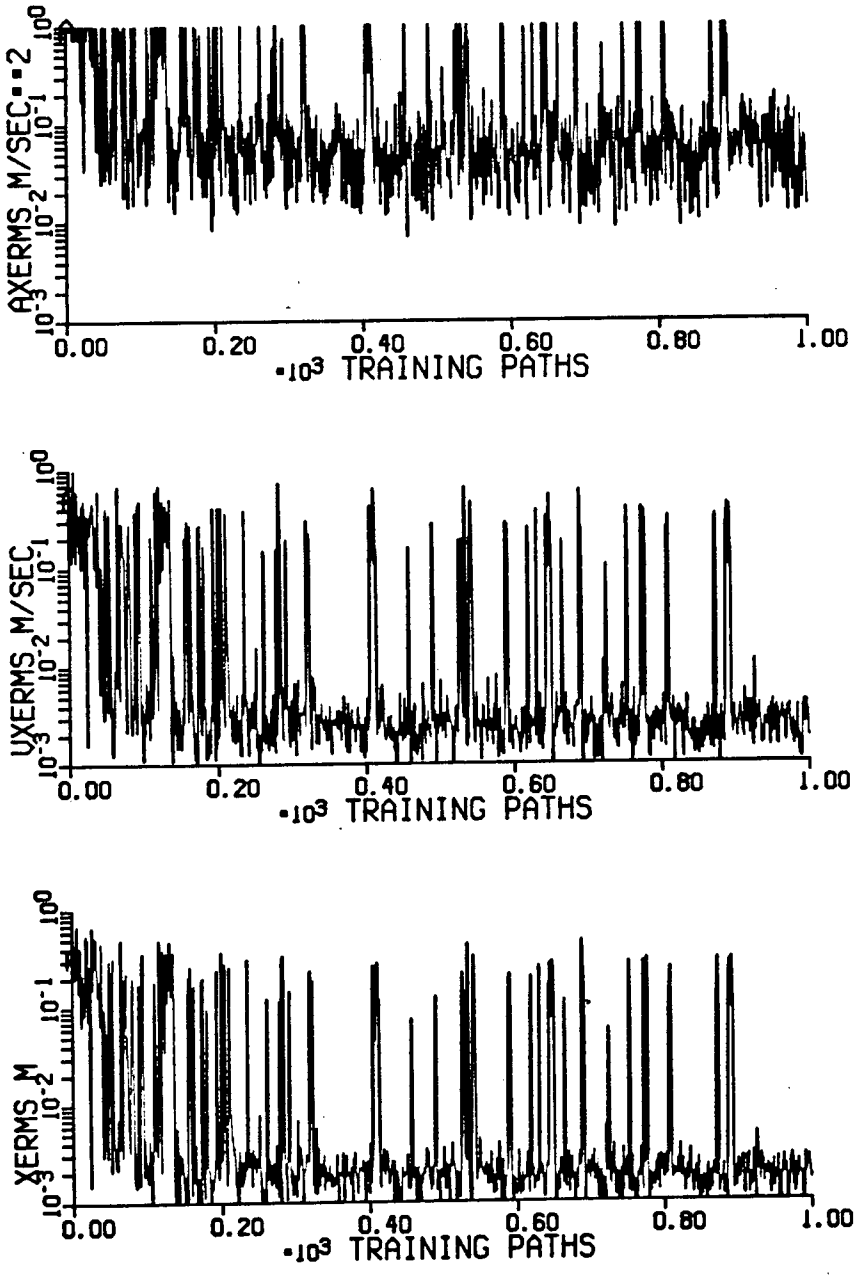


Figure 3.35 Path errors during self-learning of the Cartesian inverse dynamics

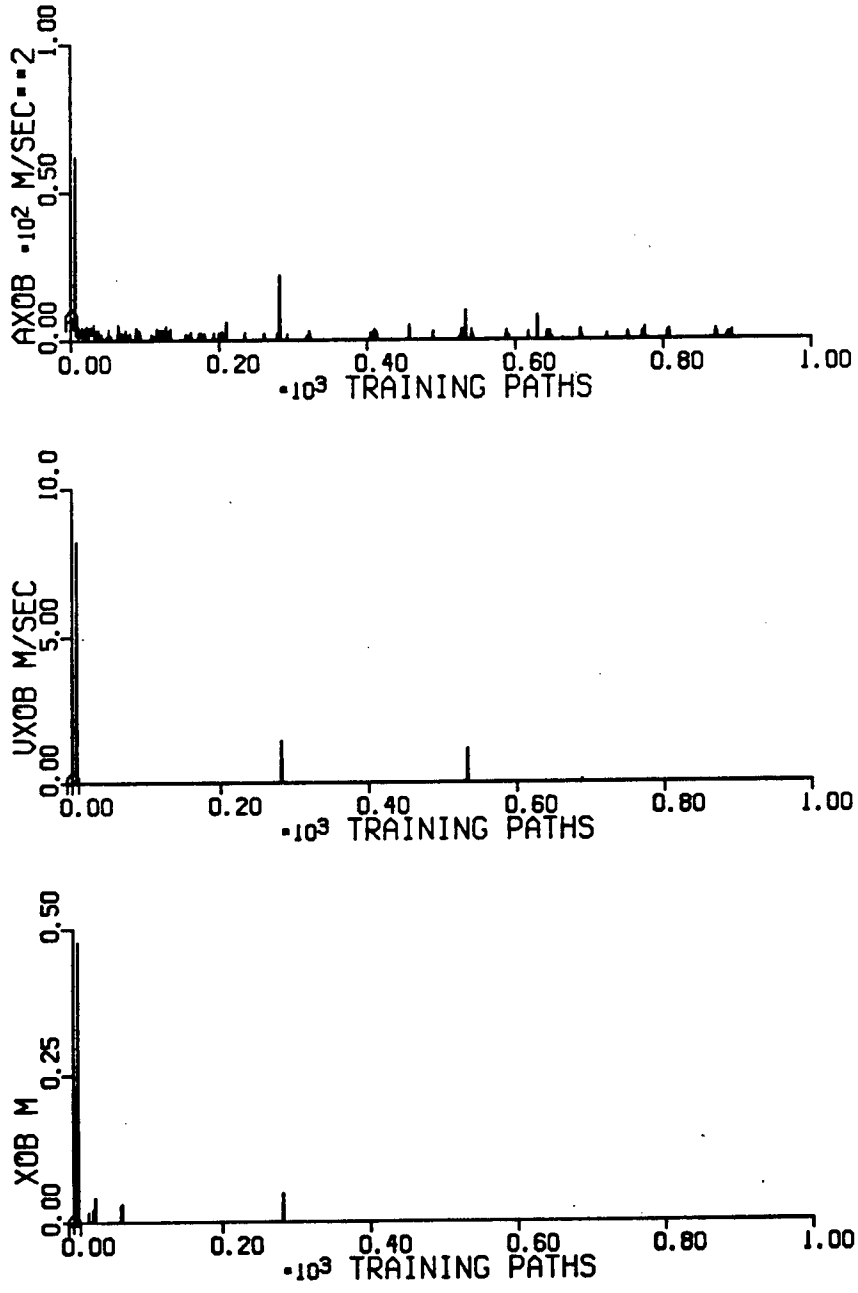


Figure 3.36 Maximum out of bounds excursions during self-learning of the Cartesian inverse dynamics

It is acceleration that is most poorly tracked and hence most frequently out of bounds. In fact, slight overshooting of steps in acceleration can exceed the bounds on acceleration and cause constraining action to be invoked which, may then result in velocity and position exceeding their bounds. This occurs because constraining action simply returns the manipulator within bounds without concern as to where. When constraining action is ceased, the large path tracking error that is likely to exist may cause a large corrective acceleration which again takes the manipulator out of bounds. Thus a slight overshooting of steps in acceleration can lead to an episode of thrashing in which velocity and position bounds are also exceeded. Many of the out of bounds excursions that occurred during the latter stages of training appeared to be of this nature. Constraints on acceleration are necessary during the initial stages of self-learning. Once the Cartesian inverse dynamics are well learned, however, it appears that more reliable control is achievable when no constraints on acceleration are imposed; one then relies on the limitation of applied torques to limit acceleration.

A better constraining technique might be to invoke constraining action that would initially damp out manipulator motion. After the manipulator is stopped, joint servoing could then be used to slowly move the manipulator towards the center of Cartesian path space, as before. Once the manipulator is observed to again be within Cartesian path space, the next training path could be begun, starting from the current manipulator position. This should avoid the protracted episodes

of thrashing that occurred using the simplistic constraining technique described previously.

As mentioned previously, self-learning of the Cartesian inverse dynamics required 1000 training paths. This corresponded to 2948 seconds (49 minutes) of self-learning. If t_{lrn1} were increased then required self-learning time would also increase.

3.8.2 CLOSED LOOP CONTROL USING THE SELF-LEARNED CARTESIAN INVERSE DYNAMICS

The coefficients of the self-learned Cartesian inverse dynamics were saved after 200, 400, 600, 800 and 1000 training paths. Simulations of closed loop control of all of the six standard paths were carried out to test the adequacy of the self-learned Cartesian inverse dynamics as learning progressed. Simulation parameters were set as follows: CLOSED=.TRUE., INVARM=.FALSE., EXACT=.FALSE., VISION=.TRUE., DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.TRUE., k_1=16, k_2=64 and t_{calc}=0.01 sec. The resulting path errors, averaged over the six standard paths, are shown in figure 3.37 as a function of the number of training paths used in self-learning of the Cartesian inverse dynamics. Figure 3.38 shows the simulation of line 1 using the final self-learned Cartesian inverse dynamics. Plots of the other simulations of line 1 are shown in appendix C. Figure 3.39 shows a plot of the simulation of circle 1 using the final self-learned Cartesian inverse dynamics.

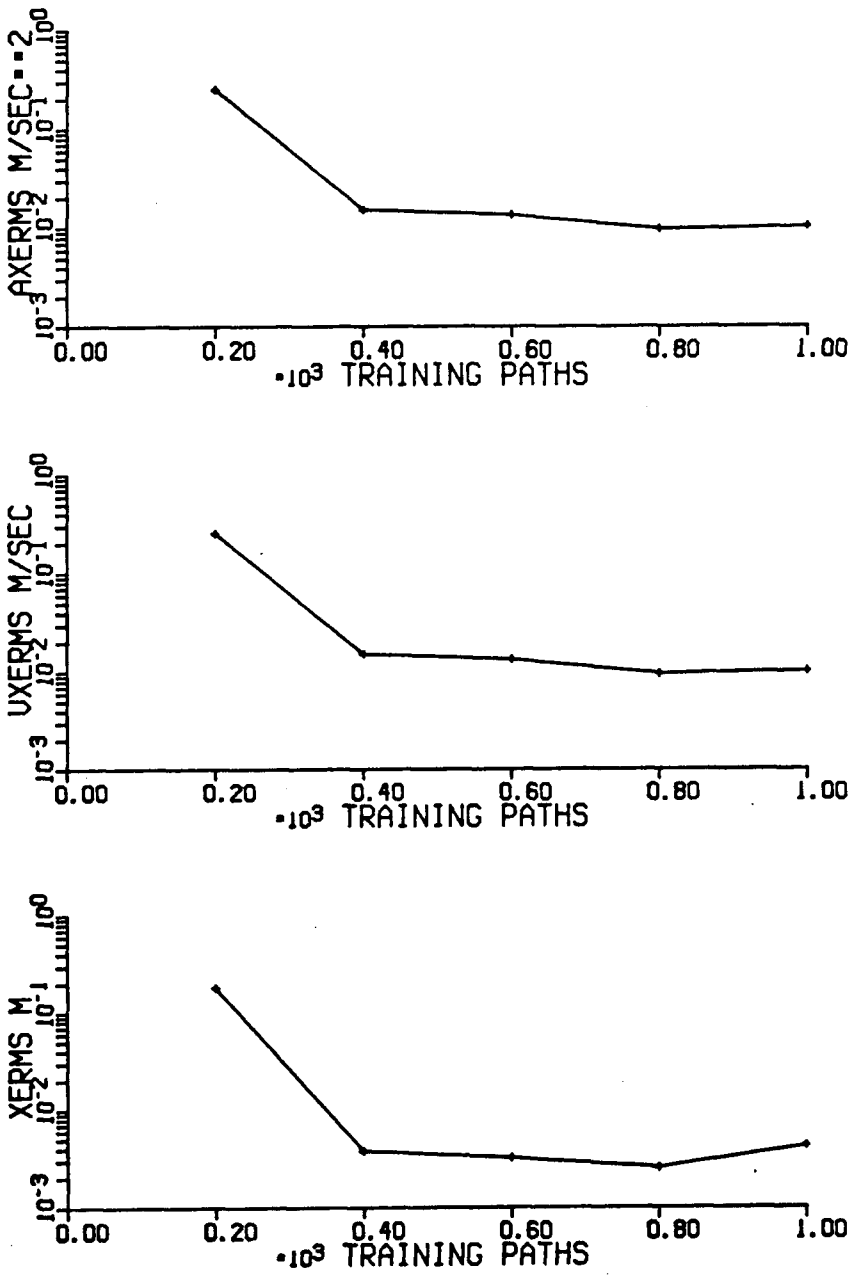


Figure 3.37 Average path errors during closed loop control of the six standard paths using the self-learned Cartesian inverse dynamics as a function of the number of training paths used.

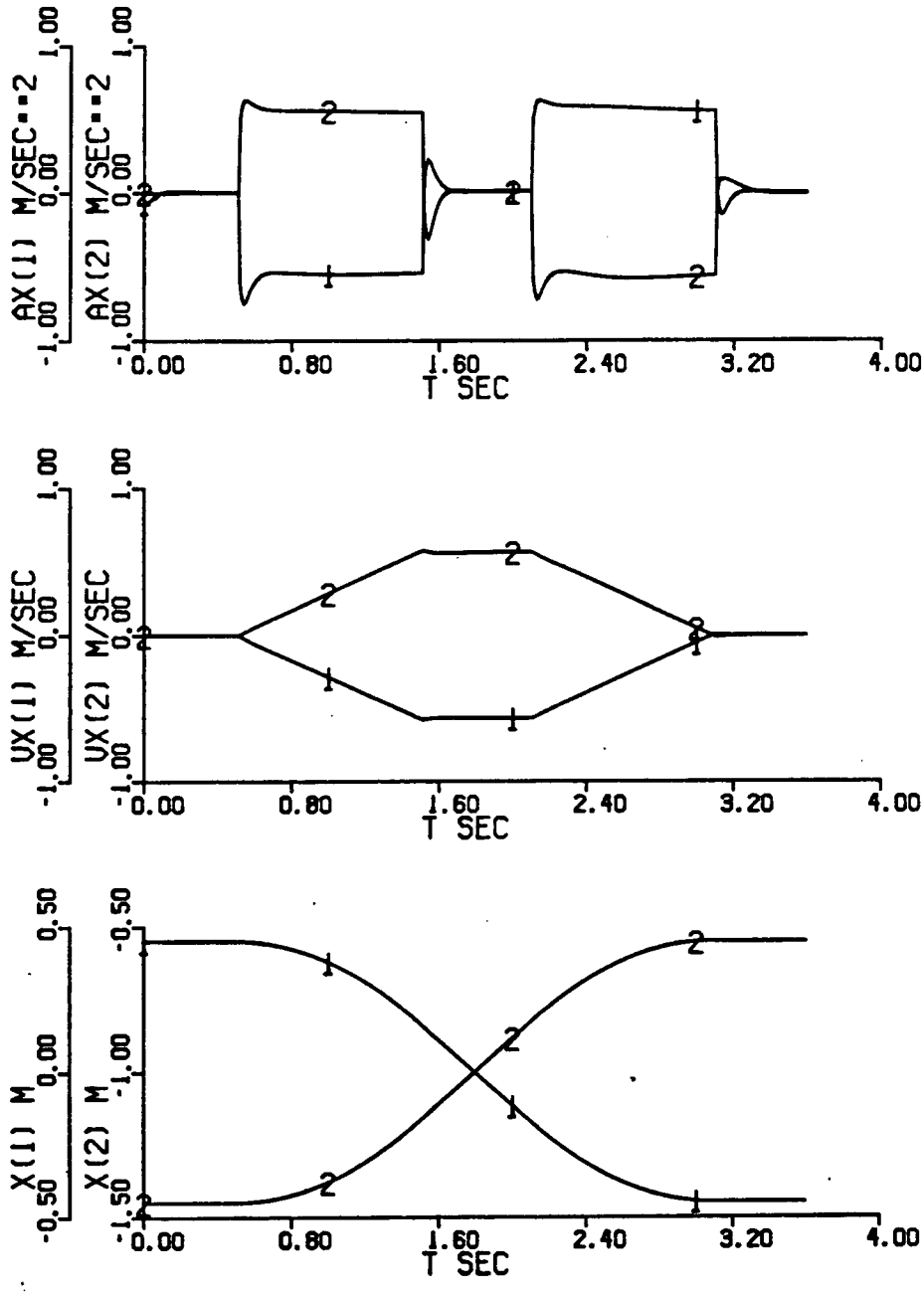


Figure 3.38 Closed loop control of line 1 using the final self-learned Cartesian inverse dynamics

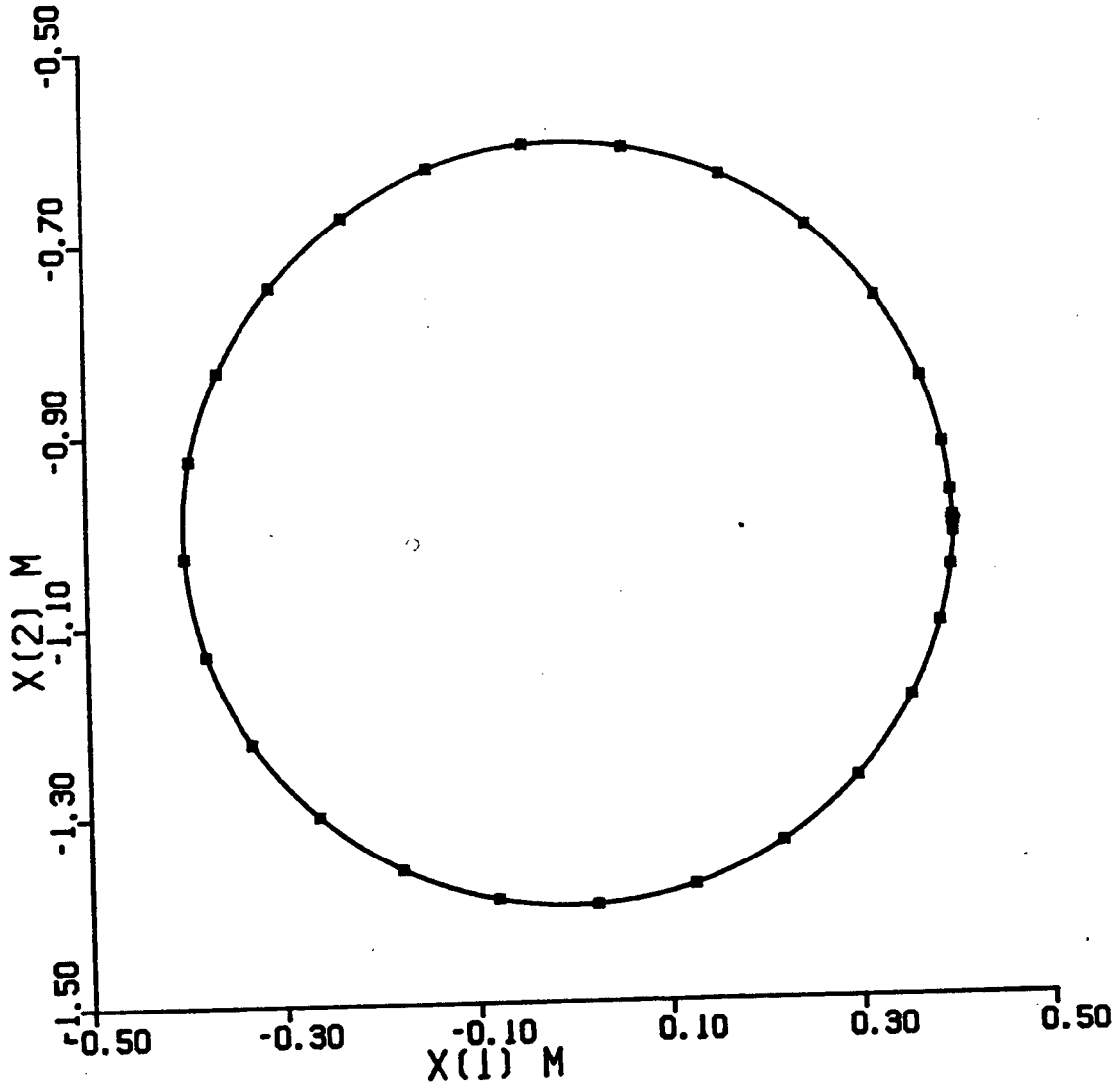


Figure 3.39 Closed loop control of circle 1 using the final self-learned Cartesian inverse dynamics

Torque estimates using the self-learned Cartesian inverse dynamics were compared to torque calculated with the analytical Cartesian inverse dynamics. Torque error, averaged over the six standard paths are shown in figure 3.40 as a function of the number of training paths used.

Path errors and torque error decrease during self-learning until minimum values are reached. After 1000 training paths it appears that these errors cannot be reduced further. The control performance after 1000 training paths thus represents the limit of what can be achieved with a 4^{th} order, self-learned sum of polynomials representation of the Cartesian inverse dynamics.

The resulting path errors for the simulations of the six standard paths using the Cartesian inverse dynamics self-learned over 1000 training paths are listed in table 3.15. Also listed in table 3.15 are the torque error and the maximum torque applied during each standard path. Performance is only slightly poorer than with standard analytical control. Position tracking errors are typically less than 3 mm.

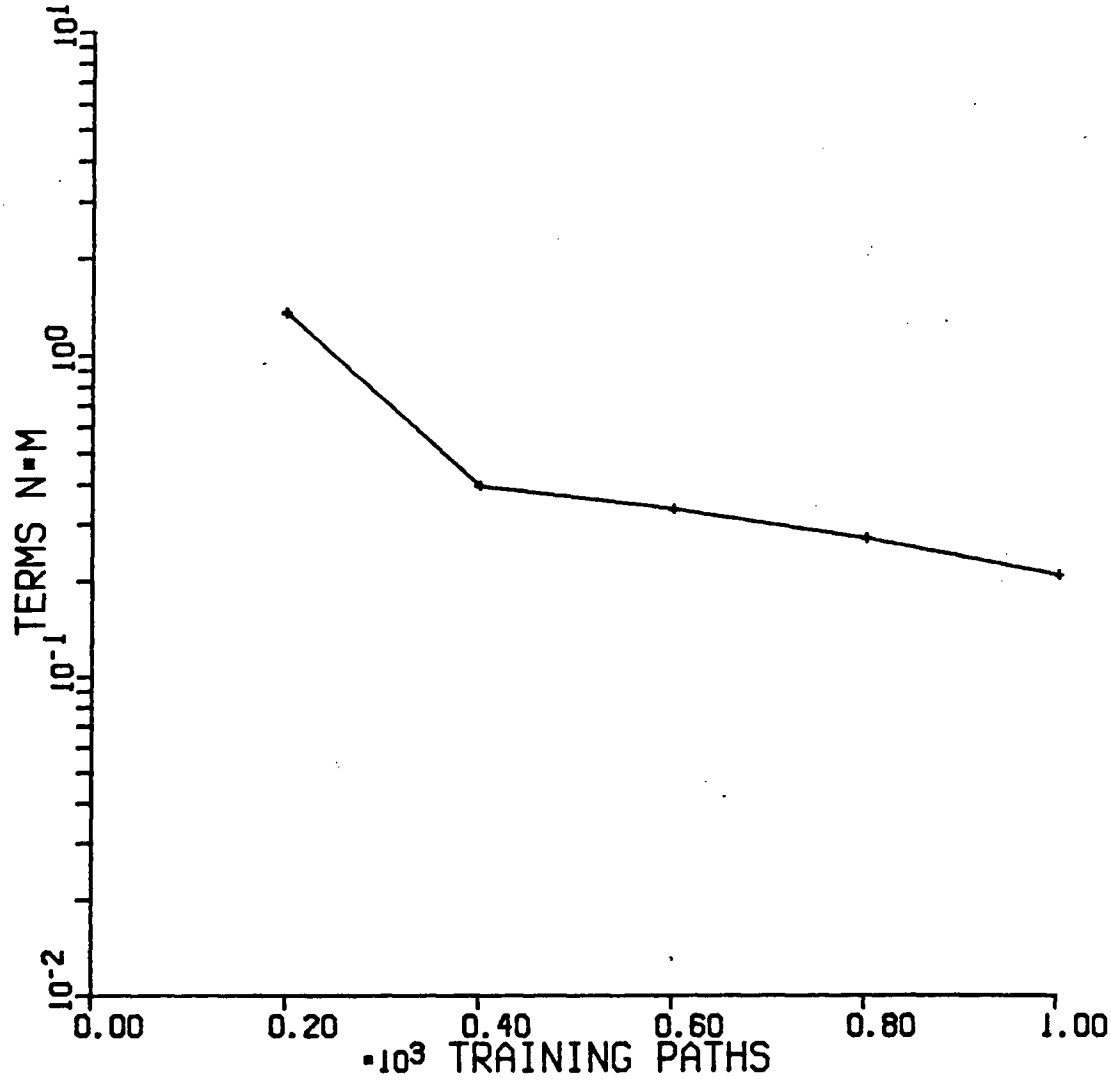


Figure 3.40 Average torque estimation error during closed loop control of the six standard paths using the self-learned Cartesian inverse dynamics as a function of the number of training paths used.

PATH	XERMS	VXERMS	AXERMS	TERMS	TRQMAX
line 1	1.60E-3	4.02E-3	1.01E-1	1.96E-1	8.27
line 2	2.15E-3	4.75E-3	1.03E-1	2.47E-1	9.41
line 3	1.47E-3	3.90E-3	1.08E-1	1.64E-1	14.0
line 4	1.74E-2	4.09E-2	2.92E-1	3.00E-1	6.64
circle 1	1.16E-3	3.85E-3	5.29E-2	1.52E-1	9.59
circle 2	1.62E-3	3.45E-3	5.26E-2	2.00E-1	13.5

Table 3.15 Path error and torque error using the final self-learned Cartesian inverse dynamics in closed loop control

3.9 SELF-LEARNING OF THE DIRECT POSITION KINEMATICS

A sum of polynomials representation of the direct position kinematics has been pre-learned and found adequate for coarse control. It remains only to find a method of learning the direct position kinematics without recourse to analysis of the manipulator. We have found that this can be done through observation of manipulator joint positions and the corresponding manipulator end point position in Cartesian coordinates as seen by the vision system.

3.9.1 A METHOD FOR SELF-LEARNING OF THE DIRECT POSITION KINEMATICS

THE METHOD

Given sufficient and varied observations of the manipulator position in joint coordinates and manipulator position in

Cartesian coordinates, one can use Interference Minimization or related methods to learn the direct position kinematics.

To learn the direct position kinematics over the joint position space given by equations (3.151) and (3.152), the same two conditions must be met as during self-learning of the Cartesian inverse dynamics: First, one must obtain observations corresponding to all regions of the position space, $-1 < z_i < 1$. Here the z_i are the normalized joint positions that act as input variables in the sum of polynomials representation of the direct position kinematics. Secondly, one must constrain the manipulator motion to be within the bounds $-1 < z_i < 1$ in order that learning can take place.

Since we wish to use the direct position kinematics only over the workspace, it is only necessary that we learn the direct position kinematics over the work space, not over the whole joint position space given by $-1 < z_i < 1$. Also, since the workspace is within the joint space given by equations (3.151) and (3.152), constraining manipulator motion within the workspace as was done during self-learning of the Cartesian inverse dynamics also serves to constrain the manipulator within the joint space $-1 < z_i < 1$. It was thus possible to perform self-learning of the direct position kinematics simultaneously with the self-learning of the Cartesian inverse dynamics. Note that, unlike the self-learned Cartesian inverse dynamics, the self-learned direct position kinematics were not used for control purposes while self-learning was taking place.

Simulations were performed of 4200 training paths during

which self-learning of the direct position kinematics was carried out. For the first 1000 training paths self-learning of the Cartesian inverse dynamics was simultaneously carried out. Thus the simulation parameters were the same as given previously. Nothing was changed for the simulation of the final 3200 training paths except that self-learning of the Cartesian inverse dynamics was disabled by setting LRNINV=.FALSE.

Learning parameters were set as follows: LRNVIS=.TRUE., t_{lrn2}=0.01 sec and Δf_{min}=0.001 m. Method 3, Interference Minimization was used. The number of input variables was v = 2 and the system order was s = 4. The input variables were those given in equations (3.151) and (3.152).

At each interval of t_{lrn2}, learning was conditionally performed. First, the observed manipulator joint positions had to be within the bounds $-1 < z_i < 1$ in order that learning could take place. Then the coefficients of the partially learned sum of polynomials representation of the direct position kinematics were used off-line to estimate the Cartesian position corresponding to the observed joint positions. These coefficients were initially zero. In self-learning of the direct position kinematics, the estimation error, Δf, is the difference between the estimated Cartesian position and the corresponding Cartesian position that was observed by the vision system at the instant that the joint positions were observed. If this estimation error was greater that Δf_{min} for either of the Cartesian coordinates then the sum of polynomials representation of the direct position kinematics function for the corresponding

Cartesian coordinate was adjusted using Interference Minimization.

The training paths that were used were the same as described previously for self-learning of the Cartesian inverse dynamics.

RESULTS

The proportion of opportunities during each path at which learning takes place is shown in figure 3.41. Unlike during self-learning of the Cartesian inverse dynamics, this proportion remains relatively constant during self-learning of the direct position kinematics. This reflects the fact that the manipulator exceeds the bounds on input variables representing normalized joint positions for the direct position kinematics, less often than it exceeds the more extensive bounds on input variables representing normalized Cartesian end point positions for the Cartesian inverse dynamics. Also, typical learned position estimation error of less than 0.001 m throughout the workspace could not be achieved with a 4^{th} order system and thus position estimation error was infrequently less than the threshold, Δf_{min} = 0.001 m. It is probable that Δf_{min} could have been increased, to say 0.01 m, without increasing the number of training paths necessary to self-learn the direct position kinematics while reducing the number of learning iterations required. Training at points where Δf_{min} was between 0.001 m and 0.01 m did little to speed convergence.

The coefficients of the self-learned direct position kinematics were saved at various points during the learning

process. Simulations of circle 1 were carried out using standard analytical control to ensure accurate tracking of the path specification. The self-learned direct position kinematics were used to obtain a view of circle 1 for each of the simulations. Figure 3.42 shows the improvement in the accuracy of path estimation as a function of the number of training paths used in self-learning of the direct position kinematics. Figure 3.43 shows the view of circle 1 using the final self-learned direct position kinematics. Plots of the view using the self-learned direct position kinematics for the other simulations of circle 1 are shown in appendix D. It can be seen that the self-learned path estimates initially become more accurate with training but are not improving after 4200 training paths. The performance after 4200 training paths is thus representative of the best accuracy that can be achieved using a self-learned, 4^{th} order sum of polynomials representation of the direct position kinematics. Table 3.16 lists the path estimation errors using the final self-learned direct position kinematics.

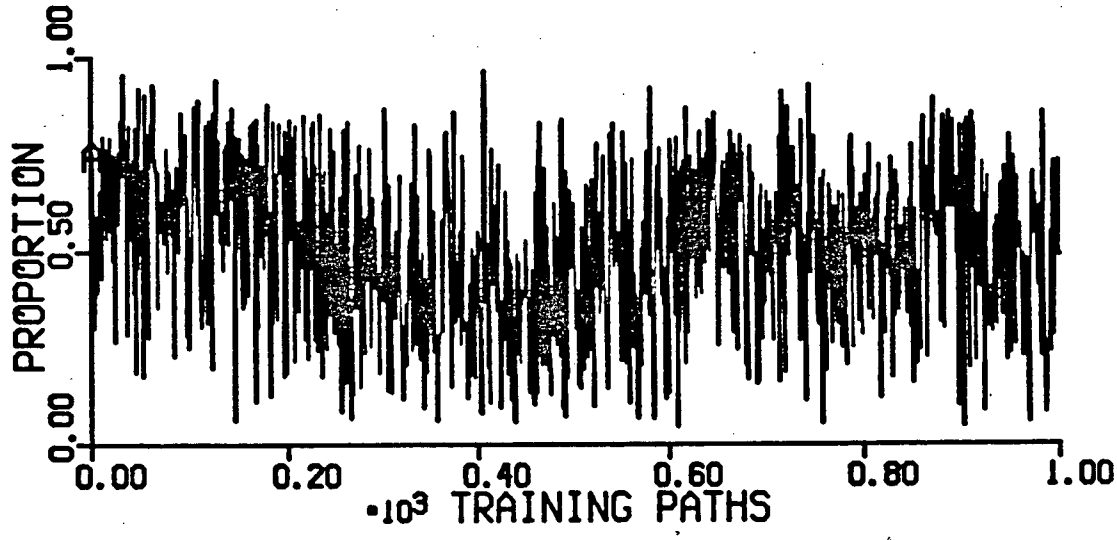


Figure 3.41 Proportion of learning opportunities at which learning took place during self-learning of the direct position kinematics

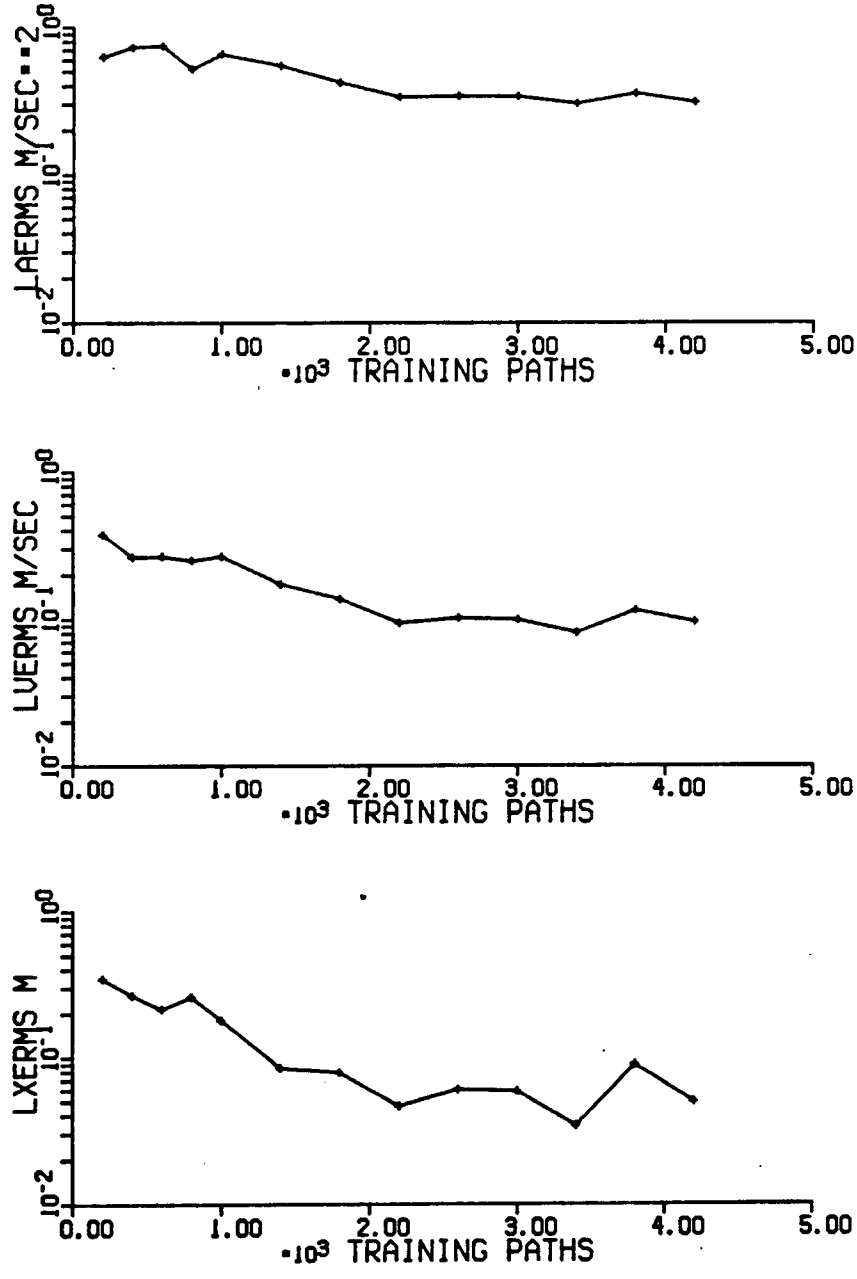


Figure 3.42 Path estimation errors during self-learning of the direct position kinematics

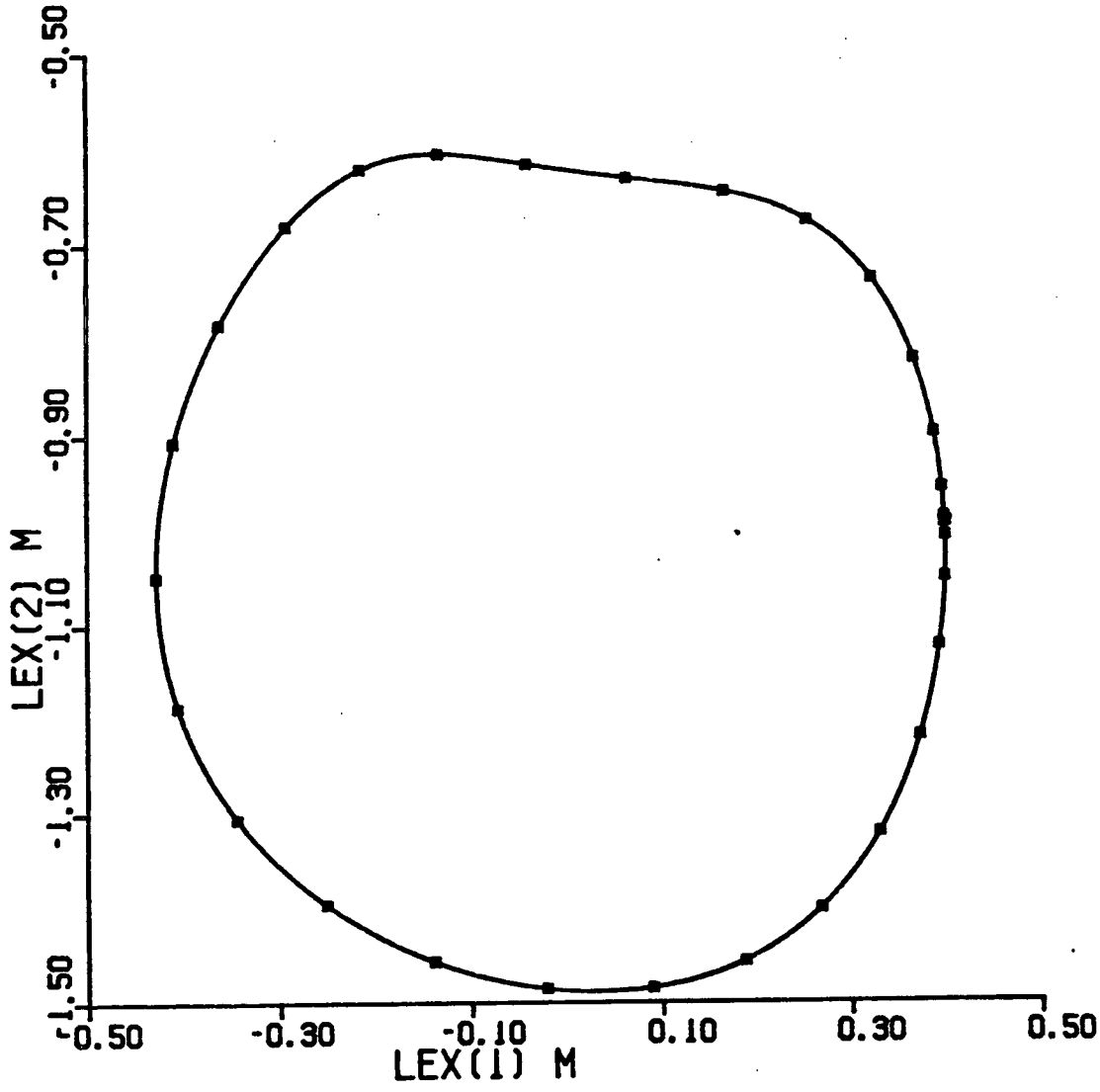


Figure 3.43 View of circle 1 using the final self-learned direct position kinematics

PATH	LXERMS	LVERMS	LAERMS
line 1	1.37E-1	1.66E-1	4.17E-1
line 2	9.96E-2	1.13E-1	2.90E-1
line 3	2.75E-2	4.65E-2	1.64E-1
line 4	7.56E-2	7.86E-2	2.52E-1
circle 1	4.97E-2	9.61E-2	3.10E-1
circle 2	5.28E-2	9.01E-2	3.02E-1

Table 3.16 Path estimation error using the final self-learned direct position kinematics

Table 3.17 lists the coefficients for the final self-learned sum of polynomials representation of the direct position kinematics alongside the derived coefficients. The correspondence is quite good except for the higher order terms. The innaccuracy in the higher order terms is not surprising as these terms are most significant near the bounds on z_i for the self-learned direct position kinematics; these bounds are outside the workspace and thus little self-learning took place near them.

Self-learning of the direct position kinematics required 4200 training paths. This corresponded to 12250 seconds (3 hours 24 minutes) of self-learning.

POLYNOMIAL TERM	X(1) COEFFICIENTS DERIVED	X(1) COEFFICIENTS SELF-LEARNED	X(2) COEFFICIENTS DERIVED	X(2) COEFFICIENTS SELF-LEARNED
1	0.0	-0.009	-1.00	-1.04
z_1	1.57	1.67	0.0	-0.006
z_1^2	0.0	-0.040	1.23	2.20
z_1^3	-0.646	-1.77	0.0	0.155
z_1^4	0.0	0.209	-0.254	-0.731
z_2	0.785	0.787	1.36	1.58
z_2z_1	-2.14	-2.60	1.23	1.62
$z_2z_1^2$	-0.969	-0.276	-1.68	-0.750
$z_2z_1^3$	0.879	1.49	-0.507	-1.41
z_2^2	-1.07	-0.879	0.617	0.277
$z_2^2z_1$	-0.969	-0.051	-0.839	-0.699
$z_2^2z_1^2$	1.32	-0.492	-0.761	-2.05
z_2^3	-0.323	-0.653	-0.559	-1.54
$z_2^3z_1$	0.879	1.45	-0.507	-0.633
z_2^4	0.220	0.982	-0.127	1.13

Table 3.17 Coefficients for the derived and final self-learned sum of polynomials representations of the direct position kinematics functions for positions x_1 and x_2

From the derivation and pre-learning of the Cartesian inverse dynamics and direct position kinematics, one would expect the direct position kinematics to be easier to learn. Several factors combine, however, to make it more difficult to self-learn a sum of polynomials representation of the direct position kinematics that are adequate for control purposes than to do likewise for the the Cartesian inverse dynamics. First, the Cartesian inverse dynamics do not need to be known very accurately to achieve good control. For feedback error correction to be reasonably effective it appears to only be necessary that sign of the differential, $\partial\tau_i/\partial\ddot{a}_i$, be known accurately throughout path space; the magnitude needs only to be roughly known. Secondly, the direct position kinematics must be very accurately known. Errors in the self-learned direct position kinematics translate directly into tracking errors and, if significant enough, obviate the cancellation of system non-linearities required for stable resolved acceleration control of the manipulator. Finally, the training paths are less effective in providing suitable training points for the self-learning of the direct position kinematics. When self-learning the Cartesian inverse dynamics, the training paths result in changes to all of the six input variables being used. Furthermore, steps in acceleration occur. This means that training points close together in time are often far apart in terms of the space represented by the six input variables. When self-learning the direct position kinematics there are only two input variables and these variables are not changing quickly. Training points

close together in time are thus close together in terms of the space represented by the two input variables. Since Interference Minimization eliminates estimation error exactly at a training point and the sum of polynomials estimate is continuous, there will be little estimation error at nearby subsequent training points. Self-learning only takes place when estimation errors occur and hence it requires more time to self-learn the direct position kinematics.

It also seems surprising that the self-learned direct position kinematics are not as accurate as the pre-learned direct position kinematics. Again, this is due to difference in the way training points are chosen. In self-learning, the final training points all occur on the final training path which represents only one region of the space. Thus the self-learned kinematics are quite accurate in the region of the last training path at the expense of increased inaccuracy at other points in the workspace. The random choosing of training points during pre-learning of the direct position kinematics does not favor one particular region of the workspace and thus results in a mean accuracy that is better. It is possible to modify self-learning so that training points are less frequent and therefore more distributed over the workspace during the final stages of self-learning. One can also achieve a similar averaging or smoothing effect during the final stages of self-learning by introducing a gain factor of less than 1 in the weight adjustment formula for Interference Minimization such that estimation errors are only reduced at training points, not

eliminated. These methods should only be applied during the final stages of self-learning as they would reduce the rate of convergence during the early stages of self-learning.

To test these methods for averaging or smoothing, an additional 400 training paths were simulated during which self-learning of the direct position kinematics was carried out. Simulation and learning parameters were the same as for training paths 1001 to 4200, except that the learning interval was changed to t_{lrn2}=0.1 sec. In addition the learning algorithm adjustment formula was modified from that of normal Interference Minimization by inclusion of a gain factor of 1/10 as follows,

$$\Delta\bar{w} = 1/10 \ \frac{\Delta f \mathbf{P}^{-1}\bar{p}}{\bar{p}^{T}\mathbf{P}^{-1}\bar{p}} \qquad (3.173)$$

After completion of this additional smoothing, simulations were carried out to determine the extent of improvement. Standard analytical control was used to simulate the six standard paths and ensure accurate tracking of the path specifications. The smoothed, final self-learned direct position kinematics were used to obtain a view of each standard path. Table 3.18 lists the path estimation errors for each path using the smoothed, final self-learned direct position kinematics. Figure 3.44 shows the corresponding view of circle 1.

Table 3.19 lists the coefficients for the final self-learned direct position kinematics after additional smoothing. As in table 3.17, the correspondence is quite good except for higher order terms.

Clearly, the additional self-learning with smoothing has

improved the accuracy of the sum of polynomials representation of the direct position kinematics. It is doubtful, though, that additional such modifications of self-learning could yield further improvements. Probably a better way to achieve improvements is to use a higher order sum of polynomials, say 6^{th} order, that can more accurately represent the direct position kinematics, without requiring special variations of self-learning.

PATH	LXERMS	LVERMS	LAERMS
line 1	5.32E-2	7.43E-2	2.73E-1
line 2	3.08E-2	4.42E-2	1.77E-1
line 3	1.16E-2	2.37E-2	9.48E-2
line 4	2.33E-2	3.37E-2	1.41E-1
circle 1	1.04E-2	3.48E-2	1.47E-1
circle 2	1.17E-2	3.42E-2	1.45E-1

Table 3.18 Path estimation error using the smoothed, final self-learned direct position kinematics

POLYNOMIAL TERM	X(1) COEFFICIENTS DERIVED	X(1) COEFFICIENTS SELF-LEARNED & SMOOTHED	X(2) COEFFICIENTS DERIVED	X(2) COEFFICIENTS SELF-LEARNED & SMOOTHED
1	0.0	0.009	-1.00	-0.993
z_1	1.57	1.64	0.0	-0.015
z_1^2	0.0	-0.186	1.23	1.11
z_1^3	-0.646	-1.61	0.0	0.234
z_1^4	0.0	0.345	-0.254	-0.437
z_2	0.785	0.809	1.36	1.39
z_2z_1	-2.14	-2.34	1.23	1.28
$z_2z_1^2$	-0.969	-0.453	-1.68	-0.799
$z_2z_1^3$	0.879	1.36	-0.507	-1.13
z_2^2	-1.07	-1.26	0.617	0.407
$z_2^2z_1$	-0.969	-0.076	-0.839	-0.794
$z_2^2z_1^2$	1.32	-0.312	-0.761	-1.84
z_2^3	-0.323	-0.568	-0.559	-1.21
$z_2^3z_1$	0.879	1.21	-0.507	-0.383
z_2^4	0.220	1.38	-0.127	1.00

Table 3.19 Coefficients for the derived and smoothed, final self-learned sum of polynomials representations of the direct position kinematics functions for positions x_1 and x_2

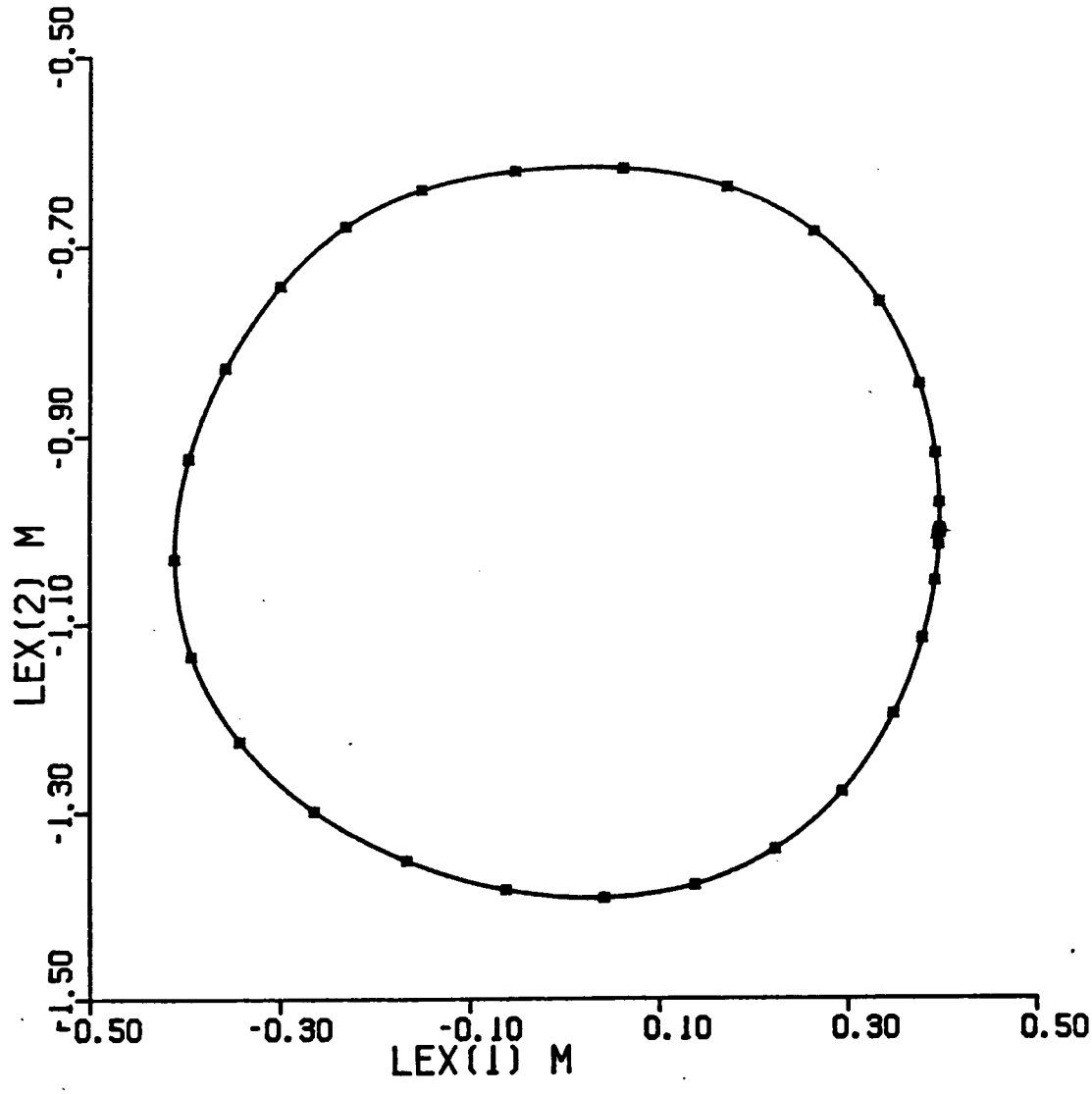


Figure 3.44 View of circle 1 using the smoothed, final self-learned direct position kinematics

3.9.2 CLOSED LOOP CONTROL USING THE SELF-LEARNED CARTESIAN INVERSE DYNAMICS AND THE SELF-LEARNED DIRECT POSITION KINEMATICS

Simulations verified that closed loop control is possible using the self-learned Cartesian inverse dynamics and self-learned direct position kinematics. We call this blind control. Simulation parameters were set as follows: CLOSED=.TRUE., INVARM=.FALSE., EXACT=.FALSE., VISION=.FALSE., DELAY=.TRUE., PRDICT=.TRUE., USEOBS=.FALSE., k_1=8, k_2=16 and t_{calc}=0.01 sec. The standard feedback gains, k_1=16 and k_2=64, were deemed too high as they resulted in instability. The best results were obtained by reducing the feedback gains to the values given above. All six standard paths were simulated. The resulting path errors are shown in table 3.20.

Tracking is worse than that achieved with the pre-learned direct kinematics, however, this is to be expected as the final self-learned direct position kinematics are less accurate than the pre-learned direct position kinematics. Path errors are of the same order of magnitude as position estimation errors, as shown previously in table 3.16. Figure 3.45 shows the resulting path as seen using the final self-learned direct position kinematics during blind control of circle 1. Control is clearly marginal as the control system is unable to make the perceived path follow the path specification that well. The actual path is shown in figure 3.46 and is somewhat worse than the perceived path, as expected.

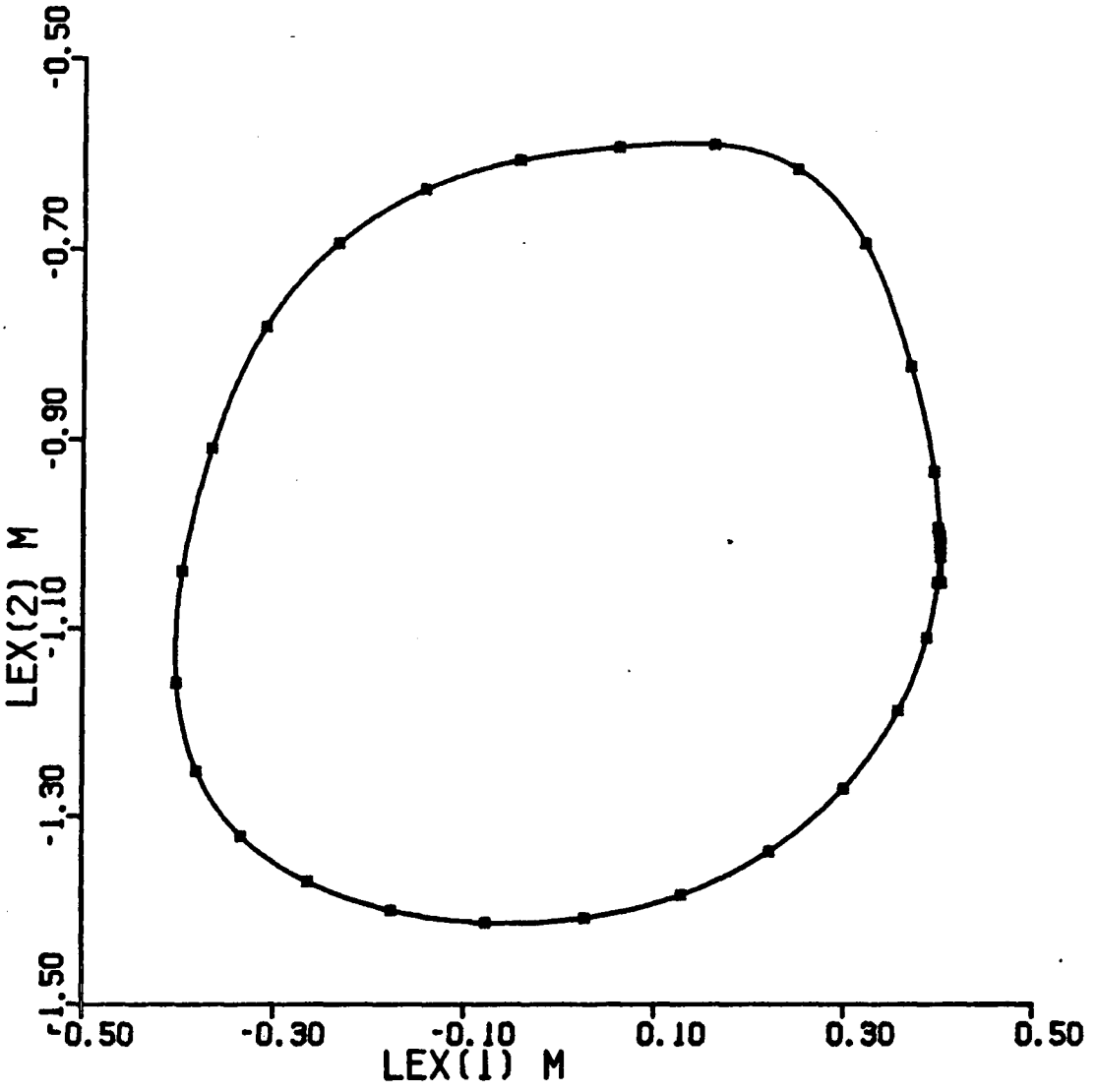


Figure 3.45 View of circle 1 using the final self-learned direct position kinematics during closed loop control of circle 1 using the final self-learned Cartesian inverse dynamics and final self-learned direct position kinematics

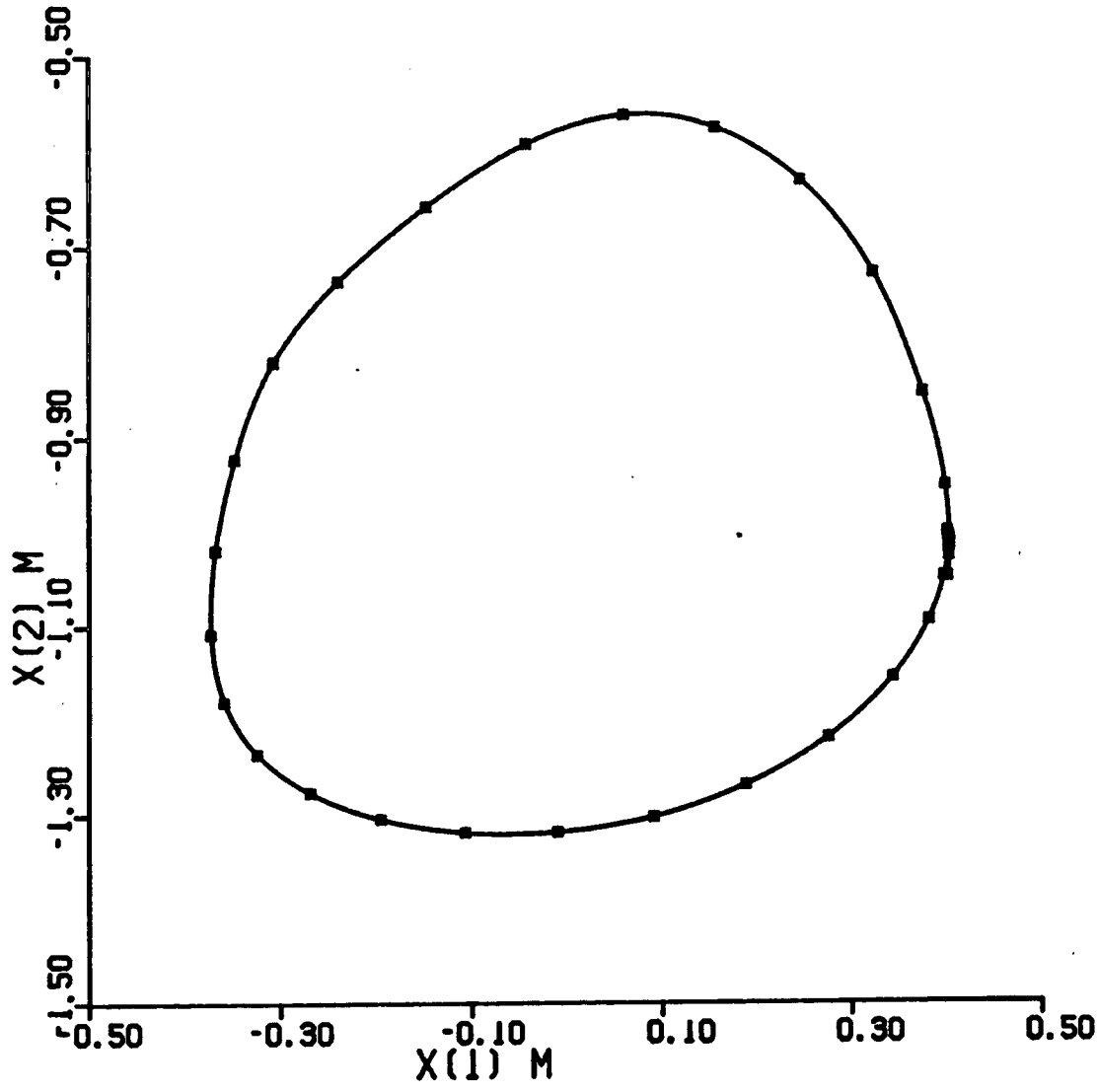


Figure 3.46 View of circle 1 during closed loop control of circle 1 using the final self-learned Cartesian inverse dynamics and final self-learned direct position kinematics

PATH	XERMS	VXERMS	AXERMS
line 1	1.37E-1	1.66E-1	4.17E-1
line 2	9.96E-2	1.13E-1	2.90E-1
line 3	2.75E-2	4.65E-2	1.64E-1
line 4	7.56E-2	7.86E-2	2.52E-1
circle 1	4.97E-2	9.61E-2	3.10E-1
circle 2	5.28E-2	9.01E-2	3.02E-1

Table 3.20 Path error using the final self-learned Cartesian inverse dynamics and final self-learned direct position kinematics in closed loop control

Additional simulations were carried out using the smoothed, final self-learned direct positions kinematics in closed loop control. Simulation parameters were not altered. All six standard paths were simulated. The resulting path errors are listed in table 3.21.

Tracking is now as good as that achieved with the pre-learned direct position kinematics. This is to be expected as the additional averaging has rendered the smoothed, final direct position kinematics to be as accurate as the pre-learned direct position kinematics. Path errors are of the same order of magnitude as position estimation errors, as given previously in table 3.18. Figure 3.47 shows the resulting path as seen using the smoothed, final self-learned direct position kinematics during blind control of circle 1. Control is much better now

than when using the final self-learned direct position kinematics without smoothing, since the control system is able to make the perceived path follow the path specification. The actual path is shown in figure 3.48 and is somewhat worse than the perceived path, as expected.

PATH	XERMS	VXERMS	AXERMS
line 1	3.83E-2	9.27E-2	4.04E-1
line 2	8.41E-2	1.18E-2	2.34E+0
line 3	1.15E-2	2.71E-2	1.41E-1
line 4	1.85E-2	3.53E-2	1.65E-1
circle 1	1.11E-2	3.86E-2	1.71E-1
circle 2	1.40E-2	4.13E-2	1.79E-1

Table 3.21 Path error using the final self-learned Cartesian inverse dynamics and smoothed, final self-learned direct position kinematics in closed loop control

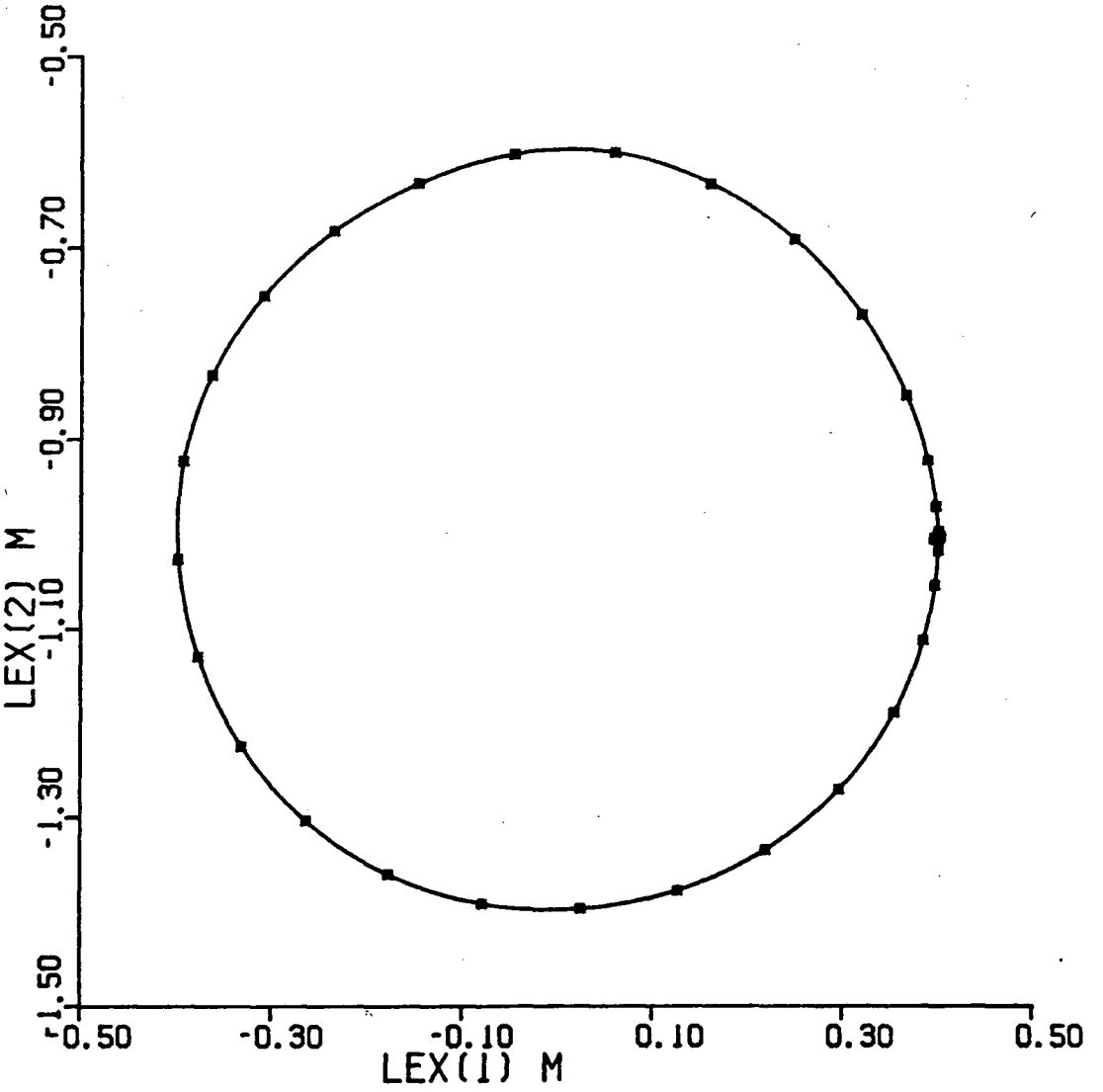


Figure 3.47 View of circle 1 using the smoothed, final self-learned direct position kinematics during closed loop control of circle 1 using the final self-learned Cartesian inverse dynamics and smoothed, final self-learned direct position kinematics

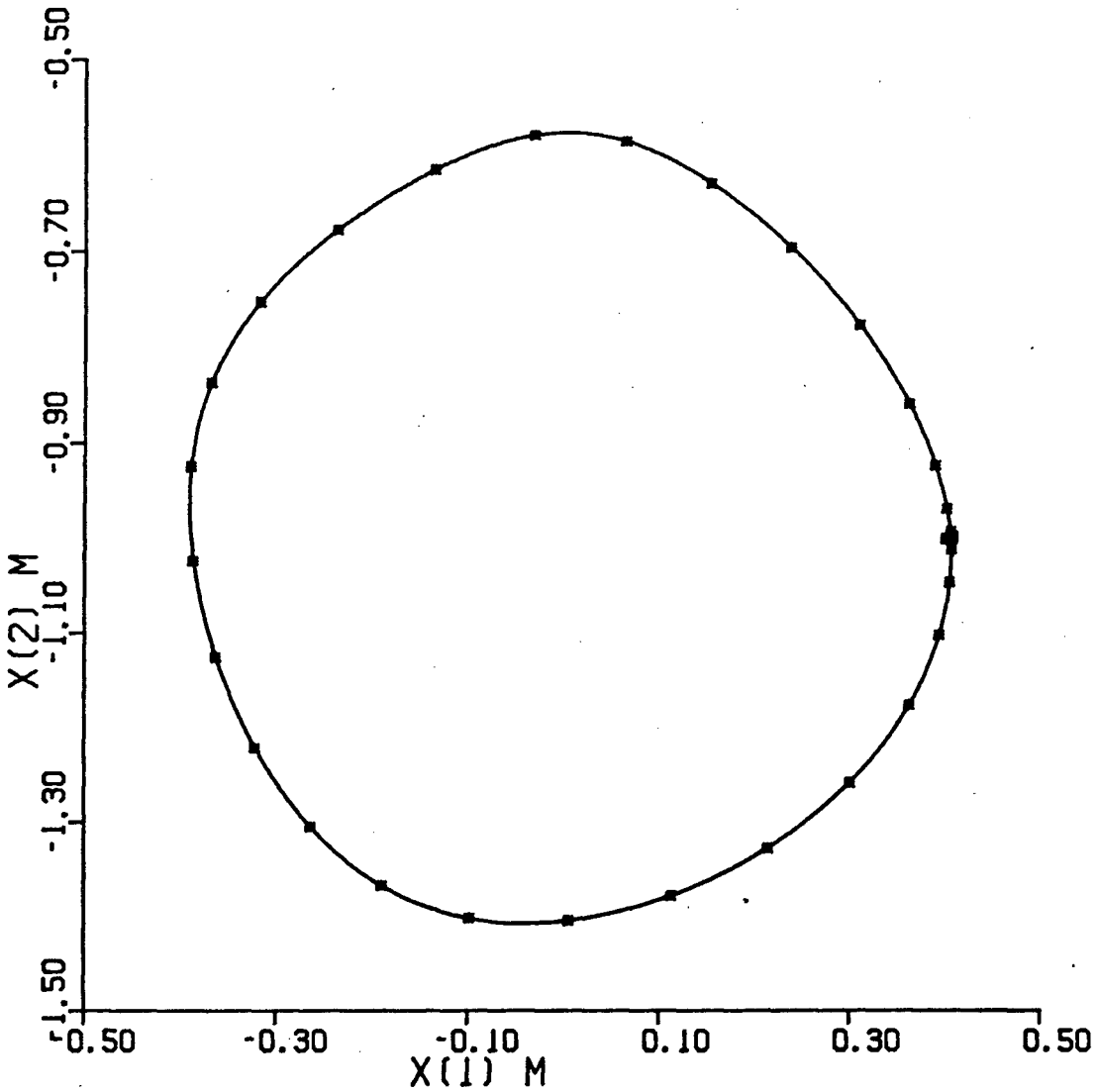


Figure 3.48 View of circle 1 during closed loop control of circle 1 using the final self-learned Cartesian inverse dynamics and smoothed, final self-learned direct position kinematics

Figure 3.49 shows the resulting path during blind control of line 1. The large accelerations that initially occur are due to the sudden perceived position error that results when one switches from the accurate vision system to the less accurate smoothed, final self-learned direct position kinematics. As was done before when using the pre-learned direct position kinematics, this glitch can be eliminated by utilizing an initial iteration of Learning Identification or Interference Minimization before switching from the vision system to the smoothed, final self-learned direct position kinematics. Figure 3.50 shows the resulting path during blind control of line 1 where this technique was enabled by setting ADJVIS=.TRUE.

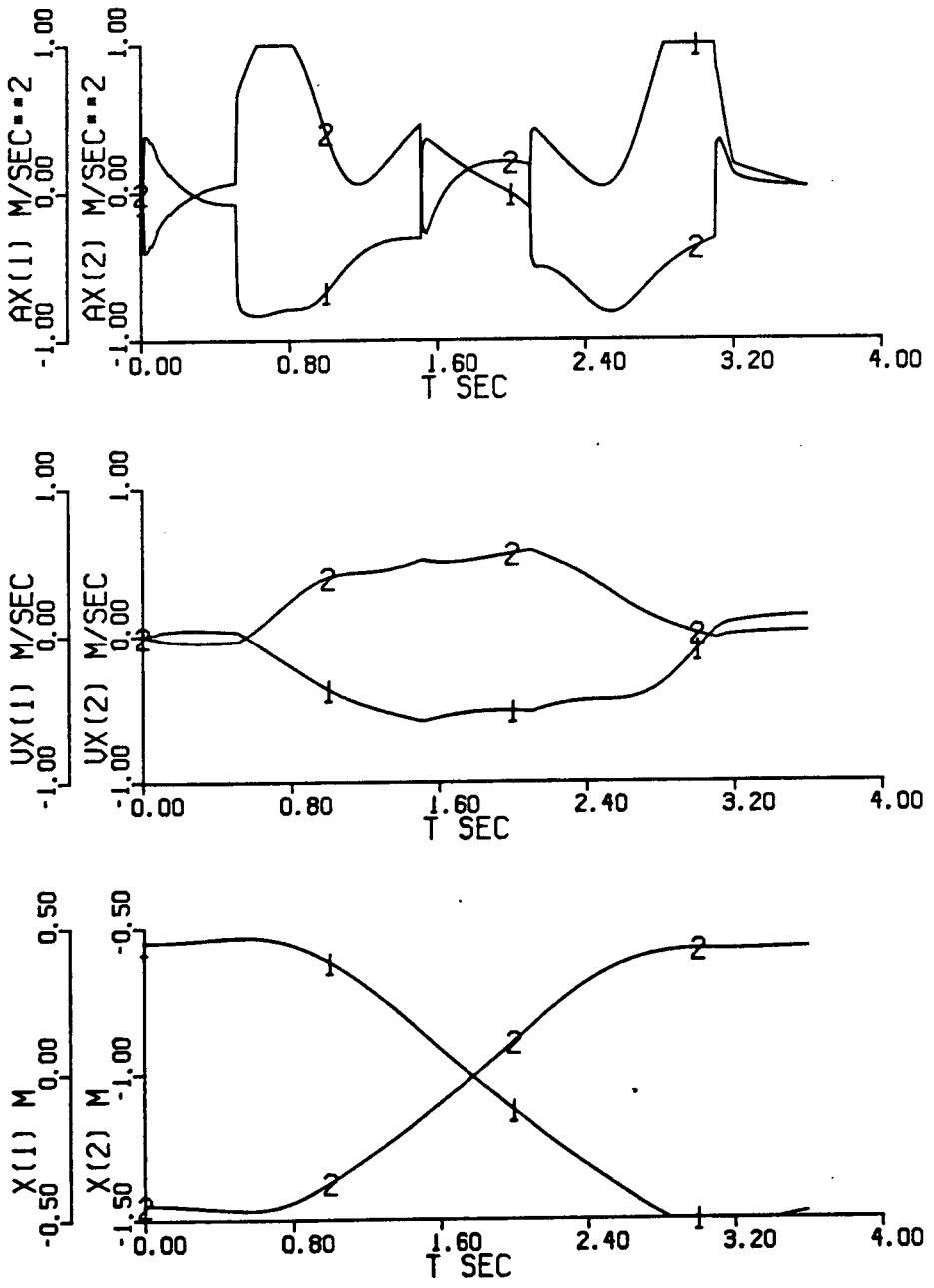


Figure 3.49 Closed loop control of line 1 using the final self-learned Cartesian inverse dynamics and smoothed, final self-learned direct position kinematics

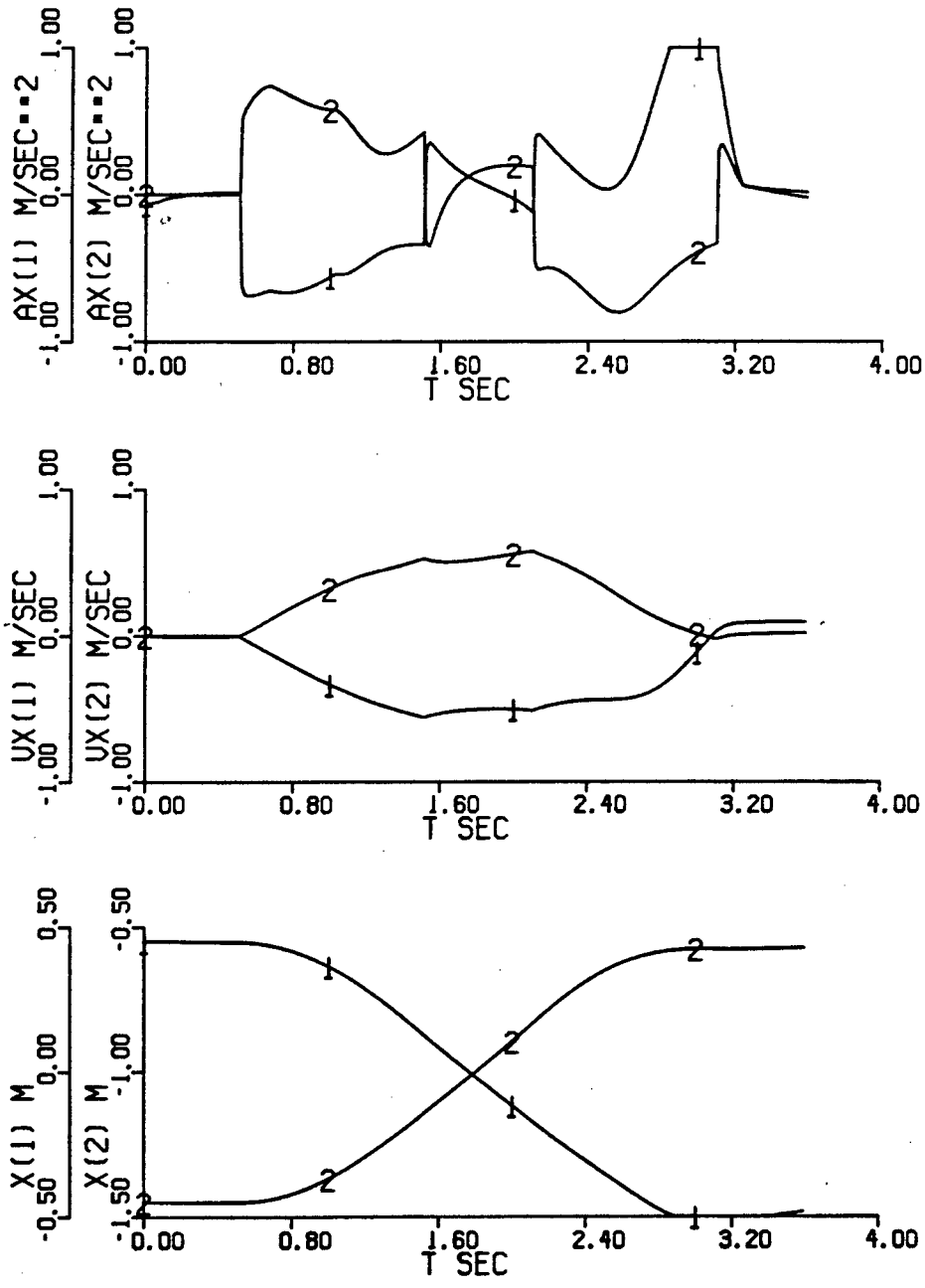


Figure 3.50 Closed loop control of line 1 using the final self-learned Cartesian inverse dynamics and smoothed, final self-learned direct position kinematics with ADJVIS=.TRUE.

These simulations show that coarse control is possible using the self-learned Cartesian inverse dynamics and self-learned direct position kinematics. Finer control appears to require only an improvement in the accuracy of the self-learned direct position kinematics. By using averaging or smoothing techniques during the final stages of self-learning, it is possible to improve the accuracy of the self-learned direct position kinematics to the point that they are equal to or better than the pre-learned direct position kinematics. Good control is possible using the smoothed, final self-learned direct position kinematics. Further improvements are likely if higher order sum of polynomials representations are used that can more accurately represent the direct position kinematics over the workspace. Since there are only two input variables when estimating the direct position kinematics, one can use high order, sum of polynomials representations without incurring the computational costs of having a large number of terms in the sums.

3.10 SUMMARY

The concept of self-learned control has been introduced and its performance demonstrated using a simulated two link manipulator. This performance has been compared to the control performance achieved using the analytical dynamics and kinematics of the two link manipulator. Control using the analytical dynamics and kinematics has been extensively tested to ensure that standard analytical control forms a realistic

benchmark against which to compare self-learned control.

Self-learning is based on the fact that it is possible to represent the non-linear, multi-variate dynamics and kinematics relationships of the two link manipualator as weighted sums of polynomials. Using Interference Minimization or related learning algorithms, it is possible to learn the weighting coefficients of the sums of polynomials and thus learn the functional relationships necessary to achieve control. Furthermore, it is possible to learn these functional relationships without recourse to analysis of the manipulator.

The inverse dynamics of the two manipulator can be well represented as a sum of polynomials over the whole of the manipulator's reach. This would seem to be true in general for manipulators as their inverse dynamics appear in a similar format [28,46]. The sum of polynomials representation can be thought of as merely replacing the various sine and cosine terms by their series equivalents and combining alike terms. This was demonstrated by derivation of a sum of polynomials representation by just such a technique. The sum of polynomials representation thus yields an efficient computational format for evaluation of the functional relationships as well as permitting the learning of these relationships with Interference Minimization.

The inverse kinematics of the two link manipulator can only be represented by sums of polynomials over a portion of the manipulator's reach at a time. The inverse kinematics of manipulators in general are difficult to obtain analytically and

are likely to contain relationships that cannot be represented by sums of polynomials over the whole of a manipulator's reach. Nevertheless we found that the inverse kinematics of the two link manipulator can be well represented as a sum of polynomials over a workspace consisting of a sizable portion of the manipulator's reach. Hence the Cartesian inverse dynamics can be self-learned over this workspace. The workspace was not specially chosen; other portions of the manipulator's reach could have been used. It would be possible to subdivide the whole manipulator space into portions and self-learn unique sum of polynomials representations for each portion. One splitting technique that may be viable is discussed in chapter 5. Another solution that may be possible is to use a more general type of learned functional estimation, perhaps based on representation consisting of a ratio of two sums of polynomials. It would appear that self-learning can be achieved for manipulators in general over a portion of the space within their reach and over the whole of their reach if a splitting technique or other more general type of learned functional estimation can be used.

The control method within which these self-learned relationships are applied is resolved acceleration control. The self-learned inverse dynamics are used to cancel the non-linearities of the manipulator dynamics yielding a combined system which is approximately linear and can be controlled using simple error correcting feedback.

Self-learned control using path specification in joint coordinates would appear to be easy to apply in general since

the inverse dynamics appear in general to be well represented by a sum of polynomials over the whole of a manipulator's reach. In this work we have implemented self-learned control using path specification in Cartesian coordinates, which is more convenient to use but more difficult to implement as it requires self-learning of the Cartesian inverse dynamics which contain the difficult to learn, inverse kinematics. Path specification in Cartesian coordinates also necessitates the use of a vision system, or its equivalent using the analytical direct position kinematics, to observe the manipulator position in Cartesian coordinates for use in error correcting feedback. The vision system or its equivalent must be very accurate as inaccuracies translate directly into path tracking errors.

The direct position kinematics of the two link manipulator can be well represented using sums of polynomials over the whole of the manipulator's reach. This appears to be possible for manipulators in general since their analytical direct position kinematics consist of sums and products of sine and cosine terms. The direct position kinematics of the two link manipulator were self-learned and used in place of the vision system in closed loop control.

Table 3.22 summarizes the path error, averaged over the six standard paths, that results when using the various control schemes. It can be seen that performance using the self-learned Cartesian inverse dynamics is quite comparable to that achieved using the analytical Cartesian inverse dynamics. Closed loop control using the self-learned Cartesian inverse dynamics in

conjunction with an accurate vision system, or its equivalent, is a very viable control technique. When the self-learned direct position kinematics are used instead of the vision system, performance is degraded, reflecting the inaccuracies in the self-learned direct position kinematics. Nevertheless it serves to demonstrate the technique. We found that an averaging or smoothing technique applied in the final stages of self-learning significantly improved the accuracy of the self-learned direct position kinematics and the resulting control performance. To achieve better performance only requires that the self-learned direct position kinematics be more accurate, as could be achieved using higher order, sum of polynomials representations.

XERMS	VXERMS	AXERMS	CONTROL SCHEME
6.48E-4	2.10E-3	8.06E-2	Standard Analytical Control
3.52E-3	5.22E-3	9.10E-2	Pre-Learned Cartesian Inverse Dynamics
4.23E-3	1.01E-2	1.18E-1	Self-Learned Cartesian Inverse Dynamics
3.37E-2	3.92E-2	2.15E-1	Pre-Learned Cartesian Inverse Dynamics & Pre-Learned Direct Position Kinematics
7.37E-2	8.60E-2	2.89E-1	Self-Learned Cartesian Inverse Dynamics & Self-Learned Direct Position Kinematics
2.96E-2	4.11E-2	5.67E-1	Self-Learned Cartesian Inverse Dynamics & Smoothed Self-Learned Direct Position Kinematics

Table 3.22 Summary of path errors using various control shemes

4 CONTRIBUTIONS OF THIS THESIS

This thesis has shown that Interference Minimization and related learning algorithms can be used to learn the non-linear, multi-variate functions describing the dynamics and kinematics of a robot manipulator. It has been shown that this learning can be done without recourse to analysis of the dynamics and kinematics of the manipulator. This is possible through learning based on observation of applied torques and corresponding manipulator motion; hence the name self-learning.

A family of learning algorithms has been studied. It has been shown that the Gradient Method and Learning Identification are simplifications of Interference Minimization, the principal learning algorithm used in this thesis. The convergence rates of Interference Minimization and these related learning algorithms have been compared for sum of polynomials estimates of various orders, having various numbers of input variables. A method has been documented for reducing the number of calculations required when implementing Interference Minimization. A new variant of Interference Minimization called Pointwise Interference Minimization has been introduced and compared with the other learning algorithms. Applications where Pointwise Interference Minimization is appropriate have been described.

The concept of self-learning has been introduced and its viability demonstrated using a simulated two link manipulator. It has been shown that the functional relationships forming the inverse dynamics, the inverse kinematics (which together form

the Cartesian inverse dynamics) and the direct position kinematics of the two link manipulator can be adequately represented by 4^{th} order sums of polynomials. This can be done over all of the two link manipulator's reach or a portion thereof sufficient to be considered a useful workspace. Self-learning of the Cartesian inverse dynamics permits resolved acceleration control to be applied using path specifications conveniently given in Cartesian coordinates. Performance using the self-learned Cartesian inverse dynamics with only a 4^{th} order representation has been found to be almost as good as performance using the analytical Cartesian inverse dynamics.

The Cartesian inverse dynamics can be learned by utilizing training points consisting of n-tuples of applied torques and resulting end point Cartesian positions, velocities and accelerations as imposed by the direct dynamics of the manipulator and observed using a vision system. The direct position kinematics can be learned by utilizing training points consisting of n-tuples of measured joint positions and corresponding end point Cartesian positions observed using a vision system. A simple technique has been demonstrated that permits the manipulator end point motion to be constrained within a desired range of Cartesian positions, velocities, and accelerations during the early stages of self-learning when the Cartesian inverse dynamics are not known well enough to control the manipulator. It has been shown that the vision system used during self-learning can be replaced with the final self-learned direct position kinematics during subsequent use of the

manipulator.

While self-learned control has only been demonstrated for a simulated two link manipulator, evidence has been presented that would lead one to believe that self-learned control is a general approach that can be applied to manipulators having more degrees of freedom using path specification in a coordinate system that is convenient for the tasks to be carried out.

Detailed suggestions for future research are presented (in the following chapter, chapter 5) covering such topics as further investigation of self-learned control (especially automatic adaptation for an unknown tool mass), hardware implementation of the required sum of polynomials estimators, and modification of the Klett Cerebellar Model (the major inspiration for this thesis) to increase its plausibility.

5 SUGGESTIONS FOR FUTURE RESEARCH

The characterization of a family of learning algorithms and formulation and demonstration of self-learned control of the two link manipulator utilizing these learning algorithms is the principal contribution of this thesis, however, perhaps just as important are certain proposals related to this work that may form the basis for future research.

5.1 FURTHER INVESTIGATION OF SELF-LEARNED CONTROL

5.1.1 CONTROL OF A MORE COMPLEX MANIPULATOR

From the outset it was hoped that self-learned control would prove a viable control scheme for complex industrial manipulators having up to 6 degrees of freedom. The two link manipulator was merely a preliminary test vehicle. We believe that no simplifying assumptions have been made that would prohibit the extension of self-learned control to a real manipulator having up to 6 degrees of freedom. The logical next step would be to attempt to apply self-learned control to a real manipulator with more than 2 degrees of freedom.

Many of the aspects of self-learning need to be studied further. It was apparent that many parameters had an effect on the rate of self-learning. One might experiment by changing feedback gains during self-learning as the optimum feedback gain in terms of minimizing out of bounds excursions, ie. instability, while minimizing path tracking error, changes as self-learning progresses. It is also apparent that the choice of

training paths, frequency of learning iterations, and forcefulness of constraining action have an effect on the learning rate. As discussed in section 3.9.1, the constraining technique used in these simulations was very simplistic and improved techniques were proposed. Investigation is warranted into changing parameters of the constraining technique used here or development of better techniques in order to minimize such quantities as maximum out of bounds excursions, risk of out of bounds excursions that might damage a real manipulator, etc. For example the constraining technique used here was found to limit out of bounds excursions quite well but there may be situations where large unacceptable out of bounds excursions occur during self-learning using this constraining technique; it has not been proven otherwise. It is also apparent that averaging techniques as discussed in section 3.9.1 can improve the mean estimation accuracy achievable with a given order sum of polynomials representation. It may also be possible to use learning algorithms based on estimation using functions other than polynomials. The costs to apply when evaluating the numerous possibilities in terms of required computing capability, self-learning time, control performance, and adequacy of constraining action, etc. will become more apparent upon application to a real manipulator.

5.1.2 ADAPTATION TO TOOL MASS

The initial intent of this thesis was to demonstrate self-learning of the inverse dynamics or the more difficult Cartesian inverse dynamics. A vision system was to be used for feedback

purposes. It soon became apparent that self-learning could be applied in other areas, for example to learn to estimate the direct position kinematics and thus replace the vision system, or free it for use in specific areas of the workspace where high resolution is required to achieve fine path tracking. Another area that invites the application of self-learning is in adaptation of control according to the tool mass.

Consider again the two link manipulator. One can imagine that the second link mass consists of the mass of the final link and the mass of the tool that the manipulator is holding,

$$m_2 = m_{link} + m_{tool} \tag{5.1}$$

The link masses along with other manipulator parameters such as link lengths, etc. were considered to be constants in functions forming the Cartesian inverse dynamics. One can also think of m_{tool} as another input variable in the Cartesian inverse dynamics,

$$\vec{\tau} = \text{cart.inv.dyn}(\ddot{\vec{x}}, \dot{\vec{x}}, \vec{x}, m_{tool}) \tag{5.2}$$

Given that m_{tool} is specified, one should be able to self-learn the Cartesian inverse dynamics for a range of tool masses. A good technique might be to initially fix the input variable, m_{tool}, and attach a tool of the corresponding mass to the manipulator while self-learning the Cartesian inverse dynamics. Once good control is achieved, one could then begin to vary the mass of the attached tool and correspondingly the variable m_{tool} and thus self-learn the Cartesian inverse dynamics for a range of tool masses. To adjust the Cartesian inverse dynamics to be appropriate for the mass of the tool in use, one need only know

m_{tool} and apply it as an input variable of the final self-learned Cartesian inverse dynamics.

A logical next step is to ask whether the tool mass can be determined simply by observing manipulator motion and applied torques. If one could discern m_{tool} based on observations of manipulator motion and applied torques then control of the manipulator could be adjusted appropriately when a tool of unknown mass is attached, a very desirable capability.

Consider the inverse dynamics of the two link manipulator as given in equations (3.51) through (3.66). One could substitute equation (5.1) into the inverse dynamics relationships and solve for m_{tool} yielding a functional relationship that we will call the tool dynamics,

$$m_{tool} = \text{tool.dyn}(\vec{\tau},\ddot{\vec{a}},\dot{\vec{a}},\vec{a}) \qquad (5.3)$$

A preliminary look suggests that this function will probably not be representable over the whole of the manipulator's reach using a sum of polynomials. It is likely, however, that it can be so represented over a portion of the manipulator's reach, such as the workspace that was chosen so that the Cartesian inverse dynamics were representable. If so, then the tool dynamics could be self-learned.

The process of generalization of self-learning need not stop here. Clearly, self-learning of the tool dynamics is quite analogous to force feedback. One merely substitutes an unknown applied force for the unknown applied tool mass. With an applied force one needs to estimate its direction as well as magnitude and hence one must learn several functional relationships.

Conceptually, though, the technique would be similar.

A wealth of possible applications for self-learning are apparent. These examples are from the field of manipulator control; applications may well exist in other fields, anywhere that learning of non-linear, multi-variate functional relationships is required. The viability of the self-learning in these applications, however, remains to be demonstrated. In some cases self-learning may theoretically be applicable but difficult to implement because of the number of calculations required when dealing with many input variables. In other cases the functions to be estimated may only be representable as sums of polynomials over narrow portions of their domains making self-learning difficult to apply with the learning algorithms given here.

5.2 IMPLEMENTATION OF SUM OF POLYNOMIALS ESTIMATORS

A principal reason for investigating self-learned control using sum of polynomials representations is the regularity of the resulting computational structure. The three learning algorithms, Interference Minimization, Learning Identification and the Gradient Method are all based on the same sum of polynomials estimator,

$$f = \bar{w}^T \bar{p} \qquad (5.4)$$

In a control application it is the evaluation of the estimate that is the time critical task. Learning of the weighting coefficients can take place off-line utilizing a method appropriate for the computing capability available and the rate

of learning desired; for example, Interference Minimization converges much faster than the Gradient Method but uses a more complex, and thus time consuming weight adjustment formula. Due to the regularity of the sum of polynomials estimator and its applicability to problems of various sizes, investigation is warranted into the implementation of sum of polynomials estimators using specially developed hardware. Preliminary work has been done for implementation using digital computing hardware and stochastic computing hardware. More work is required to demonstrate the viability of these approaches.

5.2.1 DIGITAL COMPUTER IMPLEMENTATION

In a digital computer implementation, the rate limiting factor is the number of multiplications required to evaluate the sum of polynomials. Fortunately, evaluation of the sum of polynomials does not require as many multiplications as one might assume.

Recall that, for a system of order s having v input variables, the polynomial terms are the set,

$$\{p_k(\bar{z})\} = \{ \prod_{i=0}^{v} z_i^{e_i} \} \tag{5.5}$$

where,

$$\sum_{i=0}^{v} e_i = s \ , \quad e_i \text{ an integer} \tag{5.6}$$

$$z_0 = 1 \tag{5.7}$$

The input variables and the constant 1 form the first v+1 terms. Each of the remaining higher order terms can be formed as the product of two lower order terms. The total number of terms is,

$$m = \frac{(s+v)!}{v!s!} \qquad (5.8)$$

Table 5.1 shows the number of terms for systems of various orders, s, and having various numbers of input variables, v. The number of mutiplications required to form the polynomial terms is m - (v+1).

In a manipulator control application the same polynomial terms will typically be used in the estimation of several functions. In modelling the Cartesian inverse dynamics of a manipulator with r degrees of freedom, the number of input variables will be v = 3r, corresponding to an acceleration, velocity and position for each degree of freedom. There will be r functions forming the Cartesian inverse dynamics. For such a manipulator, one pass requiring m-v-1 multiplications is necessary to form the polynomial terms and r passes requiring m multiplications and additions each are necessary to accumulate the weighted sum of these terms to complete the estimates of the r functions. For estimates of order s for each of the functions forming the Cartesian inverse dynamics, the number of multiplications required is thus,

$$cid(r,s) = \frac{(r+1)(s+3r)!}{s!(3r)!} - (3r+1) \qquad (5.9)$$

In modelling the direct position kinematics of a manipulator having r degrees of freedom, the number of input variables will be v=r, corresponding to the position for each degree of freedom. There will be r functions forming the direct position kinematics. For estimates of order s for each of the functions forming the

direct position kinematics, the number of multiplications is thus,

$$dpk(r,s) = \frac{(r+1)(s+r)!}{s!r!} - (r+1) \qquad (5.10)$$

Number of Variables	System Order 1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	6	10	15	21	28
3	4	10	20	35	56	84
4	5	15	35	70	126	210
5	6	21	56	126	252	462
6	7	28	84	210	462	924
7	8	36	120	330	792	1716
8	9	45	165	495	1287	3003
9	10	55	220	715	2002	5005
10	11	66	286	1001	3003	8008
11	12	78	364	1365	4368	12376
12	13	91	455	1820	6188	18564
13	14	105	560	2380	8568	27132
14	15	120	680	3060	11628	38760
15	16	136	816	3876	15504	54264
16	17	153	969	4845	20349	74613
17	18	171	1140	5985	26334	100947
18	19	190	1330	7315	33649	134596

Table 5.1 Number of terms, m, for sytems of various orders, s, and various numbers of input variables, v

Table 5.2 summarizes the number of multiplications required to evaluate the estimates of the Cartesian inverse dynamics for estimators of various orders, s, and having input variables corresponding to various numbers of degrees of freedom, r. Likewise, table 5.3 summarizes the number of multiplications required to evaluate the estimates of the direct position

kinematics.

Degree of Freedom	Sytem Order 1	2	3	4	5	6
1	4	16	36	66	108	164
2	14	77	245	623	1379	2765
3	30	210	870	2850	7998	20010
4	52	442	2262	9087	30927	92807
5	80	800	4880	23240	93008	325568
6	114	1311	9291	51186	235524	942153

Table 5.2 Multiplications required for estimates of various orders of the Cartesian inverse dynamics for manipulators having various degrees of freedom

Degree of Freedom	System Order 1	2	3	4	5	6
1	4	6	8	10	12	14
2	9	18	30	45	63	84
3	16	40	80	140	224	336
4	25	75	175	350	630	1050
5	36	126	336	756	1512	2772
6	49	196	588	1470	3234	6468

Figure 5.3 Multiplications required for estimates of various orders of the direct position kinematics for manipulators having various degrees of freedom

For a 4^{th} order model of a 2 degree of freedom manipulator such as the two link manipulator studied here, 623 multiplications are required. Assuming that t_{calc}=0.01 sec, this corresponds to 16 usec per multiplication. If self-learned

control can be extended to a 6 degree of freedom manipulator, 51186 multiplications would be required. For t_{calc}=0.01 sec, this corresponds to 195 nsec per multiplication, quite a high rate, but nevertheless achievable with current technology, especially if only fixed point operations are necessary. Evaluation of the direct position kinematics estimates requires far fewer multiplications and thus is easier to achieve.

The basic components of a digital computer implementation of a general purpose sum of polynomials estimator are a multiplier, an accumulator, a sequence controller to perform fetching of operands and storage of results, and memory. All portions of this system would be designed for very high speed operation; if only low speed operation were required, a standard general purpose computer could be used. This dedicated system would be supervised by a general purpose computer. Learning, which is a less time critical, but more algorithmically complex task, is assumed to be performed using one's chosen learning algorithm utilizing the supervising computer.

Figure 5.1 shows a block diagram of the proposed digital computing hardware. The digital sum of polynomials estimator would have two modes of operation. In the first mode, the set of K-G polynomial terms is generated. To begin this operation, the supervising computer places the initial terms, consisting of the constant 1 and the v input variables, into the two operand memories. It is assumed that the supervising computer has access to the operand memories and pointer memories as well registers within the sequence controller and accumulator. The higher order

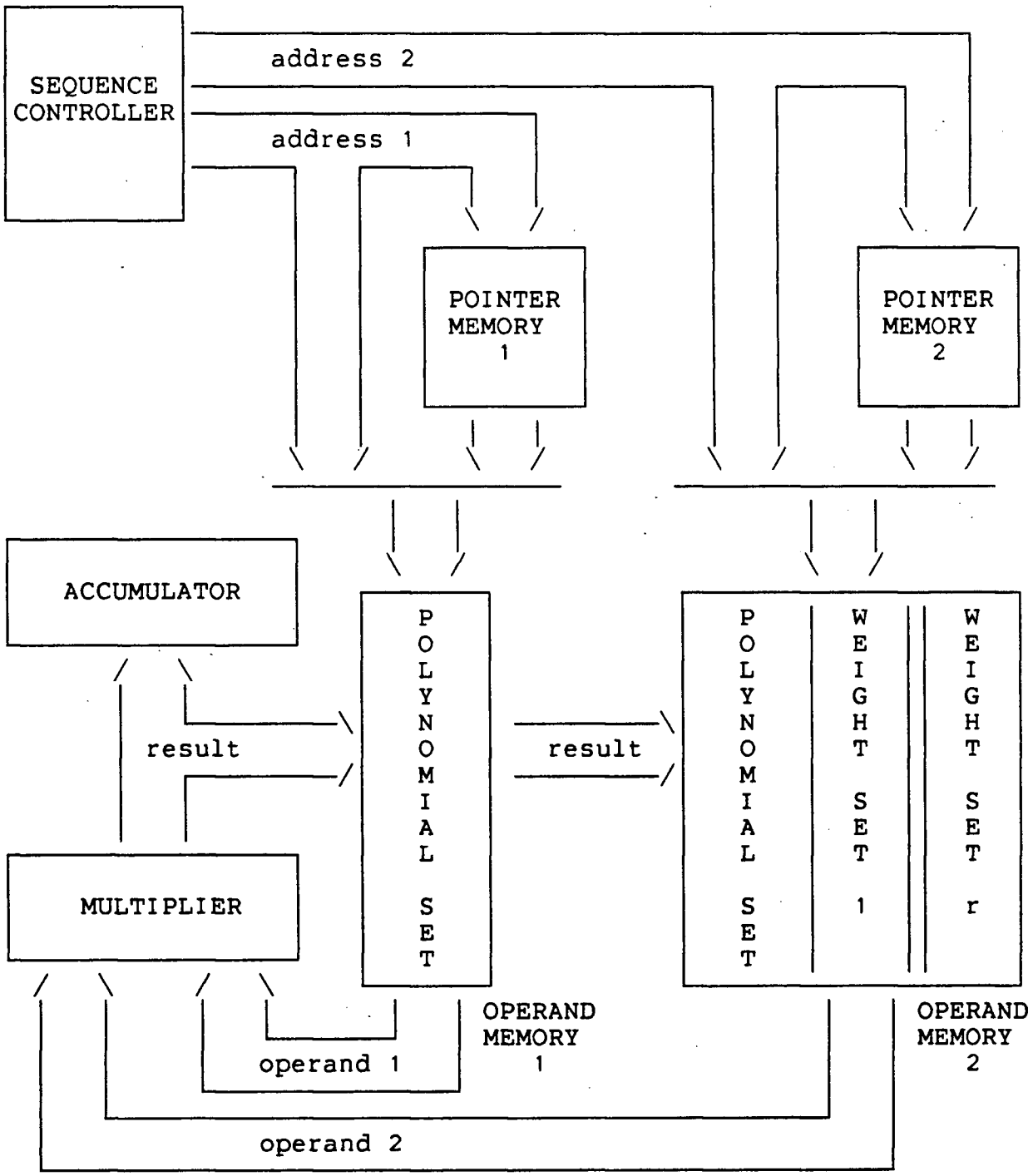


Figure 5.1 Block diagram of digital computer implementation of a sum of polynomials estimator

terms in the set of polynomials are formed in turn as products of lower order terms and appended to the set of polynomials in the two operand memories. Selection of appropriate pairs of lower order polynomial terms is performed by the pointer memories. The special pattern of addresses stored in the pointer memories is fixed for a system of given order, s, and having a specific number of input variables, v. These addresses can be generated once by the supervising computer and placed in the pointer memories. During operation, the sequence controller indexes through the pointer memories with easily generated sequential addresses. The set of polynomial terms is duplicated in each of the operand memories so that both of the required lower order polynomial term operands can be fetched at the same time. The higher order result is assumed to be written back into both operand memories simultaneously.

In the second mode of operation, each estimate is formed as a weighted sum of polynomial terms. The two operand memories are now addressed sequentially, directly by the sequence controller. Operands read from operand memory 1 consist of the polynomial terms while those read from operand memory 2 consist of the corresponding weighting coefficients for the function to be estimated. Their products are summed in the accumulator and read from the accumulator by the supervising computer at the completion of each pass. Several such passes are made using a different weight set each time until estimates have been evaluated for all r functions. Note that the number of words of memory required using this approach is,

$$mem(r,s) = \frac{(r+4)(s+2r)!}{s!r!} \qquad (5.11)$$

A key factor in determining the viability of this approach for very high speed operation is the precision required in the computations. In our simulations of self-learned control, approximately 7 decimal digits of precision were used. Our self-learned estimates of the Cartesian inverse dynamics, however, were only accurate representations of the analytical functions to about 2 or 3 decimal digits and yet were adequate for control purposes. Input variables are normalized between -1 and 1, and the weighting coefficients could also be normalized. It would thus appear that fixed point binary operations with a precision of perhaps 8 to 10 binary digits would be sufficient for control purposes. Such fixed point operations should be achievable at the rate required using current technology. If higher precision or floating point representation were found to be necessary, it would be much more difficult to implement the digital sum of polynomials estimator.

5.2.2 STOCHASTIC COMPUTER IMPLEMENTATION

It may also be possible to implement a sum of polynomials estimator using stochastic computing techniques. The stochastic computer [19,48] is distinct form either the analog or digital computer as numbers are represented by the probability of occurrence of a 1 in the bit sequences on the data lines permitting multiplication to be performed by a single gate.

For example, consider a situation in which the numbers A and B, which are between 0 and M, are represented as the

probabilities of the occurence of a 1 on their respective data lines,

$$Prob(A=1) = A/M \tag{5.12}$$

$$Prob(B=1) = B/M \tag{5.13}$$

If these signals are fed into an AND gate then its output, C, will be a 1 whenever both inputs are 1's. The probability of the output being a 1 is,

$$Prob(C=1) = Prob(A=1 \text{ and } B=1) \tag{5.14}$$

If the probabilities on the input lines A and B are independent then,

$$Prob(C=1) = Prob(A=1) \cdot Prob(B=1) \tag{5.15}$$

and the number represented on the output line is the normalized product of A and B,

$$C = AB/M^2 \tag{5.16}$$

This is a unipolar form of representation. A bipolar form of representation is also possible which permits signed multiplication using a single EXCLUSIVE OR gate [19]. Normalized summation can also be performed with similarly simple circuitry.

Adequately random input signals for a stochastic computer can be formed using a pseudorandom sequence generator and suitable logic circuitry [19]. This is done by repeatedly comparing an N bit binary representation of the input signal with an N bit binary random number obtained from a pseudorandom sequence generator. The successive results of comparison form a Bernoulli sequence whose probability of a 1 corresponds to the input signal.

We have shown that adequately random input signals can also

be obtained by randomization of pulse-rate-encoded signals using jitter stages [23].

Simple hardware and parallel operation make stochastic multiplication and summation a potential parallel processing technique. Its simplicity makes is attractive for VLSI implementation. A principal drawback, however, is that multiplications and summations may be very slow.

Consider the example given in figure 5.2. Products are formed of two pairs of inputs and these products are then summed (more correctly averaged). Assuming that the probabilities of a 1 at all inputs are independent, the probability of a 1 at the output G is,

$$Prob(G=1) = [Prob(A=1)\cdot Prob(B=1) + Prob(D=1)\cdot Prob(E=1)]/2 \quad (5.17)$$

This structure could be extended to implement a sum of polynomials estimator. To make use of the result at the output it is necessary to estimate this probability by estimating the mean proportion of 1's that occur at the output. For example, if N samples are taken of the output and O of these samples are 1's, an estimate of the probability of the output being a 1 is,

$$MEAN = O/N \quad (5.18)$$

Since the output is a Bernoulli sequence, the standard deviation in this estimate of the probability is,

$$SDEV = [Prob(G=1)(1-Prob(G=1))/N]^{1/2} \quad (5.19)$$

To halve the standard deviation in one's estimate of the output probability one must quadruple the number of observations of the output. Thus while stochastic computing may be comparatively

fast for low accuracy computations involving many input variables, it becomes very slow if high accuracy computations are required.

As discussed in section 5.2.1, it appears that the accuracy of estimates required to achieve adequate control is not very great. Thus stochastic computer implementation of a sum of polynomials estimator may be feasible. Note that, as when using fixed point arithmetic in a digital computer, narrow range restrictions are imposed on all variables at all points within a stochastic computer. Further investigation is required to determine the feasibility of such an approach.

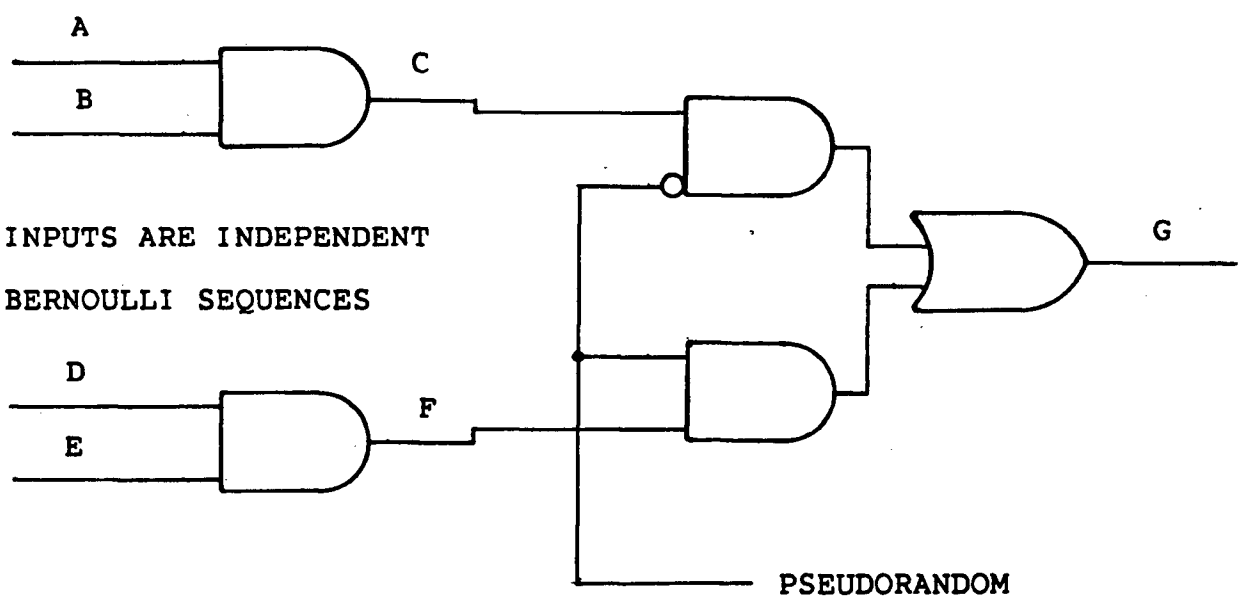


Figure 5.2 Stochastic computer circuit for summing the products of two pairs of input variables

5.3 MODIFICATION OF THE KLETT CEREBELLAR MODEL

The starting point for this work was the Klett Cerebellar Model. During the course of this research, several insights have occurred about ways of modifying the Cerebellar Model to make it more plausible as a model of the mammalian cerebellum.

5.3.1 THE KLETT CEREBELLAR MODEL

Klett [26] models the mammalian cerebellum as a sum of polynomials estimator using orthogonal polynomials. Figure 5.3 shows a block diagram of the cerebellar system. The Cerebellar Model includes only those neural pathways that are highlighted. These are the principal pathways.

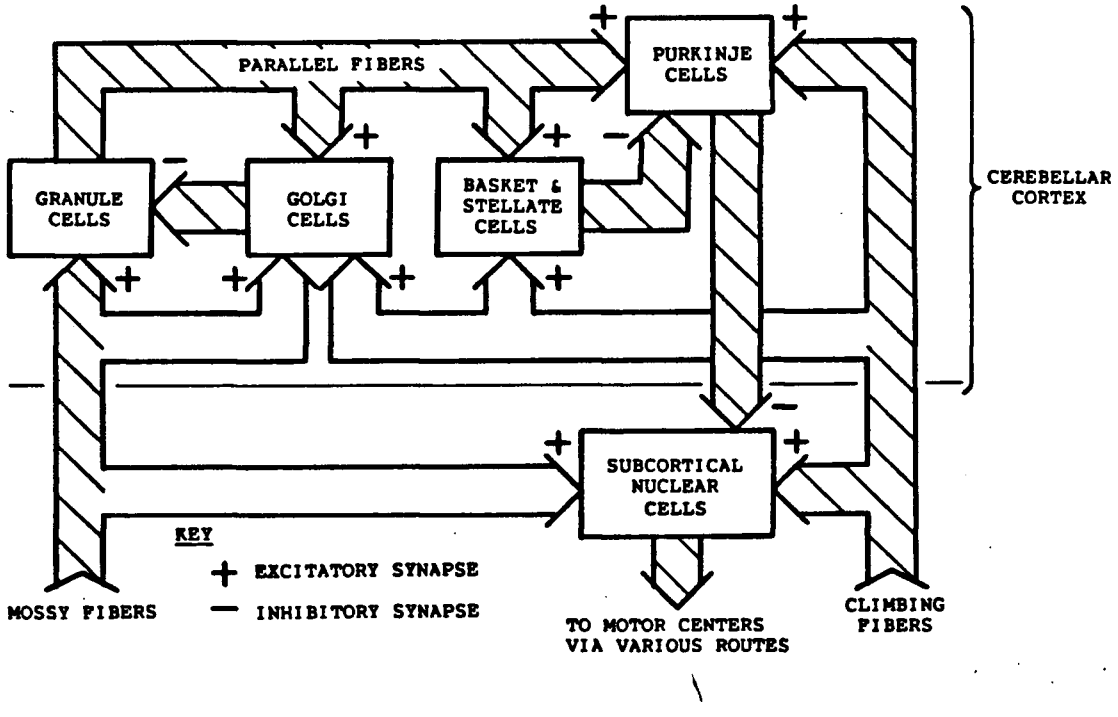


Figure 5.3 Block diagram of cerebellar system

A block diagram of the Cerebellar Model is shown in figure 5.4. Purkinje cells are excited by the parallel fibers of the granule cells and inhibited by the basket and stellate cells which are also excited by parallel fibers. The net effect would be equivalent if Purkinje cells could be excited or inhibited by the parallel fibers thus eliminating the need for basket and stellate cells. Purkinje cells and their associated basket and stellate cells are thus modelled as weighted summation points having either positive or negative weighting coefficients,

$$f = \sum_k w_k q_k = \bar{w}^T \bar{q} \qquad (5.20)$$

It is through modification of these weighting coefficients by corrective climbing fiber activity that learning is assumed to take place,

$$\Delta \bar{w} = u \Delta f \bar{q} \qquad (5.21)$$

Note that such learning is performed based on only locally available information in each Purkinje cell.

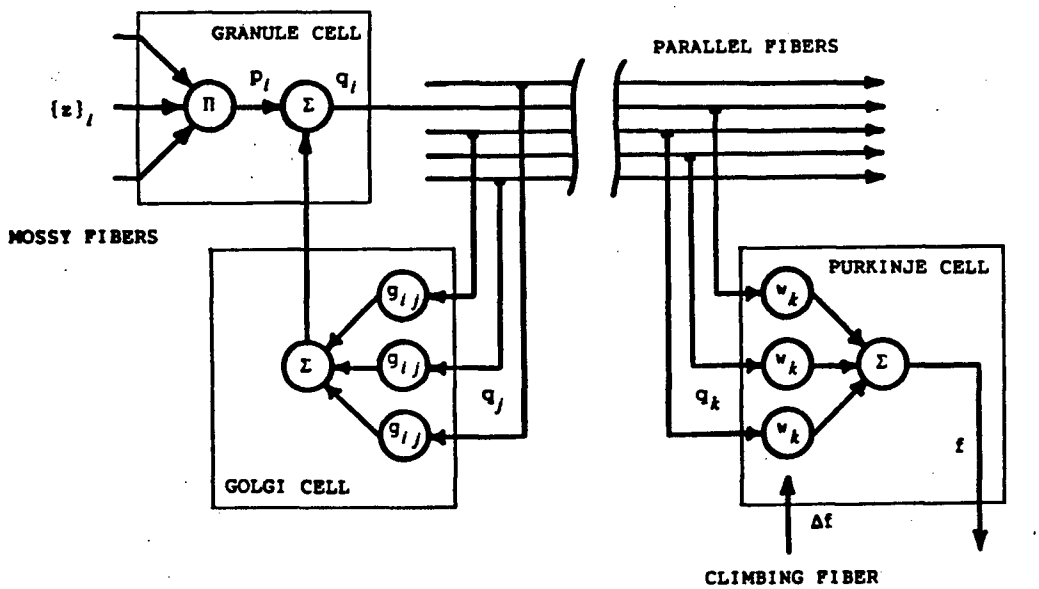


Figure 5.4 Block diagram of the Klett Cerebellar Model

Granule cells are excited by the Mossy fibers. The Granule cells are modelled as forming the product of the variables represented by Mossy fiber activity. This product is summed with the negative, inhibitory Golgi Cell activity and the result output as Parallel fiber activity. The Golgi cells are themselves excited by parallel fiber activity. The Granule cell - Golgi cell network is modelled as a negtive feedback network,

$$\bar{q} = \bar{p} - G\bar{q} \tag{5.22}$$

Steady state Parallel fiber activity is thus,

$$\bar{q} = (I+G)^{-1}\bar{p} = Q\bar{p} \tag{5.23}$$

If the Parallel fiber - Golgi cell synaptic weights represented by the matrix G are chosen appropriately then the the matrix Q becomes an orthogonalization matrix. This results in improved learning performance as shown previously in chapter 2. Such orthogonalization results if,

$$Q = [\int_S \bar{p}\bar{p}^T \, \delta S]^{-1/2} \tag{5.24}$$

This requires that G be chosen as,

$$G = [\int_S \bar{p}\bar{p}^T \, \delta S]^{1/2} - I \tag{5.25}$$

In the Klett Cerebellar Model, these special Parallel fiber - Golgi cell synaptic weights are assumed, not learned.

There is evidence that multiplication and summation required here can be performed by neurons and neural networks [26]. It is certainly possible to perform multiplication or summation of pulse-rate-encoded variables with simple digital logic gates that are analogous to neurons using stochastic computing techniques [19,48].

5.3.2 LEARNED ORTHOGONALIZATION

A principal criticism of the Klett Cerebellar Model is that it assumes the special synaptic weights in the Granule cell - Golgi cell network. This makes the Cerebellar Model rather implausible as it would require an enormous amount of information to genetically specify these weights.

To counter this argument we have found that orthogonalization is not neccesary to permit learning to take place. Without orthogonalization, the Cerebellar Model becomes equivalent to the Gradient Method outlined in chapter 2. The Gradient Method is a less optimal learning algorithm in terms of convergence rates. Nevertheless, it is capable of learning accurate sum of polynomials estimates, given enough training.

Furthermore, we have found that it is possible to learn the orthogonalization matrix Q in a manner that could conceivably be performed in the cerebellum. The learning can be done using information that is locally available in the Granule cells and Golgi cells.

Figure 5.5a shows a schematic of the Granule cell - Golgi cell network showing those locations where amplification could occur. Figure 5.5b shows the same schematic but with amplifications lumped as much as possible. With this model the Parallel fiber activity is given by,

$$q_i = a_i (b_i p_i - \sum_{j \neq i} g_{ij} q_j) \qquad (5.26)$$

Note that an individual Granule cell is assumed to not synapse with those Golgi cells that it is being inhibited by.

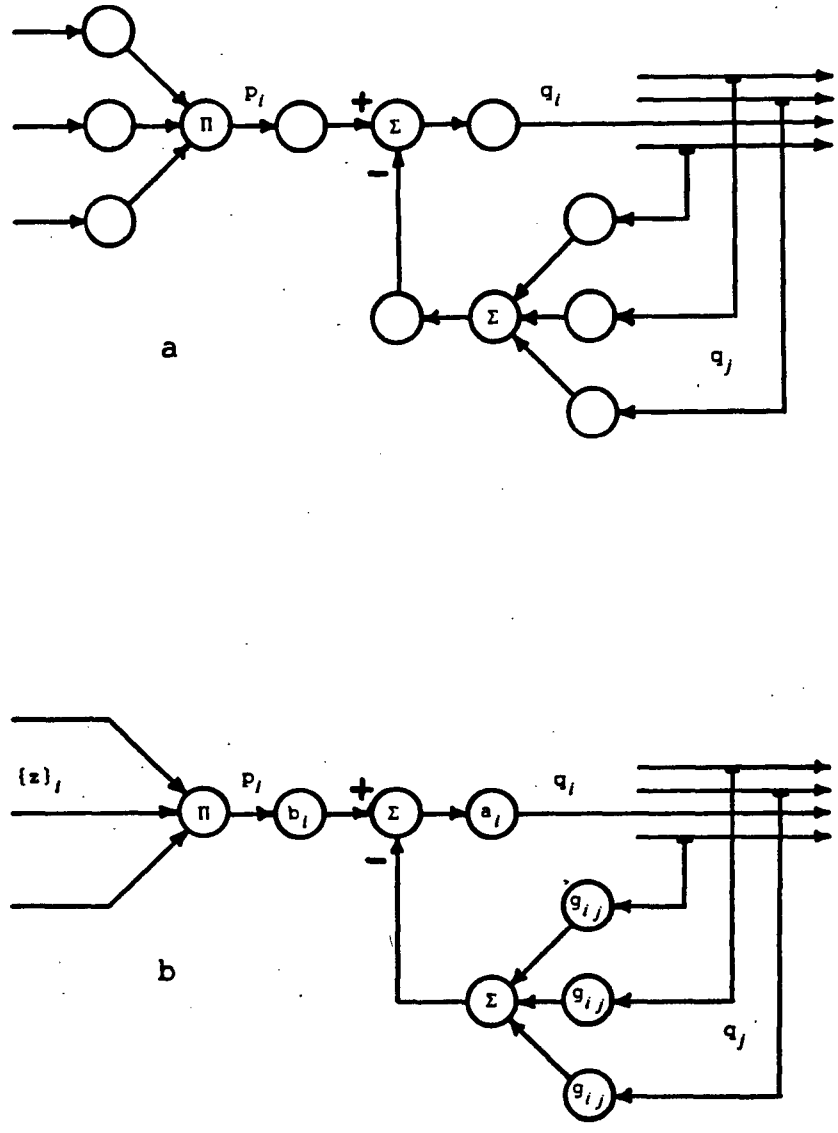


Figure 5.5 Schematic of Granule cell - Golgi cell network
a) showing those locations where amplification could occur
b) showing lumped amplifications

Expressing this in matrix form yields,

$$\bar{q} = \mathbf{A}(\mathbf{B}\bar{p} - \mathbf{G}\bar{q}) \tag{5.27}$$

A and **B** are diagonal matrices and **G** is anti-diagonal (ie. the diagonal elements are zero). Solving for $\bar{q}$ in terms of $\bar{p}$ yields,

$$\bar{q} = (\mathbf{I}+\mathbf{AG})^{-1}\mathbf{AB}\bar{p} \tag{5.28}$$

To ensure that $\bar{q}$ is an orthonormal basis set, the Pointwise Cerebellar Model discussed in chapter 2 is used. We thus desire that,

$$\bar{q} = \mathbf{E}_n\bar{p} \tag{5.29}$$

where,

$$\mathbf{E}_n = \mathbf{D}_n^{-1/2} \tag{5.30}$$

$$\mathbf{D}_n = 1/n \sum_{i=0}^{n-1} \bar{p}_i \bar{p}_i^{\,T} \tag{5.31}$$

Therefore we must ensure that,

$$(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n)^{-1}\mathbf{A}_n\mathbf{B}_n = \mathbf{E}_n = \mathbf{D}_n^{-1/2} \tag{5.32}$$

or equivalently,

$$\mathbf{B}_n^{-1}\mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) = \mathbf{D}_n^{1/2} \tag{5.33}$$

An adjustment scheme has been found that meets these requirements while using only locally available information. After the n^{th} adjustment, the elements of the matrices $\mathbf{G}_n$, $\mathbf{A}_n$ and $\mathbf{B}_n$ are as follows;

$$g_{nij} = 1/n \sum_{k=0}^{n-1} p_{ki} q_{kj} \tag{5.34}$$

$$a_{nij} = 0 \tag{5.35}$$

$$b_{nij} = 0 \tag{5.36}$$

for $i \neq j$, and,

$$g_{nij} = 0 \tag{5.37}$$

$$a_{nij} = [1/n \sum_{k=0}^{n-1} p_{ki} q_{kj}]^{-1} \tag{5.38}$$

$$b_{nij} = 1 \tag{5.39}$$

for $i=j$. Note that we are assuming that the gains b_{nij} are fixed, not learned. In matrix notation the Golgi feedback matrix is thus as follows,

$$\mathbf{G}_n = 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{q}_k^T - \mathbf{A}_n^{-1} \tag{5.40}$$

It can be confirmed that such a choice of gains results in the desired matrices by showing that equation (5.33) is satisfied. First we can remove the matrix $\mathbf{B}_n^{-1}$ from equation (5.33) as it is equal to the identity matrix.

$$\mathbf{B}_n^{-1}\mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) = \mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) \tag{5.41}$$

Then this result can be simplified.

$$\mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) = \mathbf{A}_n^{-1} + [1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{q}_k^T] - \mathbf{A}_n^{-1} \tag{5.42}$$

$$\mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) = 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{q}_k^T \tag{5.43}$$

Substituting in equation (5.28) yields,

$$\mathbf{A}_n^{-1}(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n) = 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{p}_k^T [(\mathbf{I}+\mathbf{A}_n\mathbf{G}_n)^{-1}\mathbf{A}_n]^T \tag{5.44}$$

Let us assume that the matrices $\mathbf{A}_n$ and $\mathbf{G}_n$ converge such that they appear as constants (to the degree of precision that we are concerned with) after a finite number of adjustments. If we consider equation (5.44) as $n \to \infty$, we can assume that the matrices $\mathbf{A}_n$ and $\mathbf{G}_n$ have fixed values for all terms of the summation without introducing significant error into the sum. The matrices

in the summation of (5.44) can thus be factored out.

$$\mathbf{A}_\infty^{-1}(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty) = [\ 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{p}_k^T\ ][(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)^{-1}\mathbf{A}_\infty]^T \qquad (5.45)$$

Rearrangement then yields,

$$\mathbf{A}_\infty^{-1}(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)[\mathbf{A}_\infty^{-1}(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)]^T = 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{p}_k^T \qquad (5.46)$$

The matrix $\mathbf{A}_\infty$ is symmetrical. It is not obvious that the matrix $(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)$ is also symmetrical. Let us assume that $(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)$ is symmetrical. Then the transpose in the left hand side of (5.46) can be removed yielding,

$$[\mathbf{A}_\infty^{-1}(\mathbf{I} + \mathbf{A}_\infty\mathbf{G}_\infty)]^2 = 1/n \sum_{k=0}^{n-1} \bar{p}_k \bar{p}_k^T \qquad (5.47)$$

Thus in the limit we have,

$$[\mathbf{A}_\infty^{-1}(\mathbf{I} + \mathbf{A}_\infty\mathbf{G}_\infty)]^2 = \mathbf{D}_\infty \qquad (5.48)$$

$$\mathbf{A}_\infty^{-1}(\mathbf{I} + \mathbf{A}_\infty\mathbf{G}_\infty) = \mathbf{D}_\infty^{1/2} = \mathbf{E}_\infty^{-1/2} \qquad (5.49)$$

Recalling from chapter 2 that $\mathbf{E}_\infty$ equals a scaler multiple of $\mathbf{Q}$, it is apparent that the result is as desired.

We have not actually been able to prove that this adjustment scheme will work as we had to assume convergence of the matrices $\mathbf{A}_n$ and $\mathbf{G}_n$, and we had to assume that $(\mathbf{I}+\mathbf{A}_\infty\mathbf{G}_\infty)$ was symmetrical. If these assumptions are true, then it is possible to learn orthogonalization.

In the cerebellum it does not appear to be possible to perform the adjustments of the gains as shown in equations (5.34) through (5.39). A more plausible hypothesis is that the gains are adjusted by exponential averaging where,

$$g_{ij}(k) = (1-\epsilon)g_{ij}(k-1) + \epsilon p_i(k)q_j(k) \qquad (5.50)$$

for $i \neq j$, and,

$$a_{ij}(k) = (1-\epsilon)a_{ij}(k-1) + \epsilon p_i(k)q_j(k) \qquad (5.51)$$

for $i=j$, and the other adjustment relationships are as before. ϵ is a small positive gain factor. These adjustments might occur continually but with such a small gain factor, ϵ, that adaptation is slow, resulting in behavior analogous to a low pass filter. These adjustments might also be triggered to occur when learning is taking place at the Purkinje cells as indicated by Climbing fiber activity.

Through simulation we have found that this proposed method for learned orthogonalization will work with either true averaging or exponential averaging. Succesive values of the input variables z_i used to generate the polynomial terms in $\bar{p}$ were chosen randomly. It has also been found to work regardless of whether the polynomial terms in $\bar{p}$ are strictly positive or not. It thus may prove a useful method of orthogonalizing signals, quite apart from its validity as a model of part of the cerebellum.

Several problems remain in defending this scheme as a cerebellar model. First, using such learned orthogonalization, the resulting vector, $\bar{q}$, is not strictly positive. Signals carried on the parallel fibers of the cerebellum are unipolar in nature, albeit that they may effectively represent positive and negative values over a finite range through offset by a positive bias. A more plausible orthogonalization scheme would be one that ensures that $\bar{q} - (1,\ldots,1)$ is orthogonal. Secondly, it is not clear that all Golgi cells synapse with mossy fibers such

that each Golgi cell has access to the signal p_i upon which adjustment of Golgi feedback gains are based in this model.

A more plausible model in terms of having gain adjustments being based on locally accessible signals is one where the gains are adjusted as follows,

$$g_{nij} = 1/n \sum_{k=0}^{n-1} q_{kj} \tag{5.52}$$

$$a_{nij} = 0 \tag{5.53}$$

$$b_{nij} = 0 \tag{5.54}$$

for $i \neq j$, and,

$$g_{nij} = 0 \tag{5.55}$$

$$a_{nij} = [1/n \sum_{k=0}^{n-1} q_{kj}]^{-1} \tag{5.56}$$

$$b_{nij} = [1/n \sum_{k=0}^{n-1} p_{ki}]^{-1} \tag{5.57}$$

Now the gains are assuredly adjusted on the basis of locally available information. This method cannot be proven to work, however, precisely because the q_i are not strictly positive. It has been found not to work in simulations, for the same reason. If a method can be found for offsetting $\bar{q}$ to ensure that the q_i are strictly positive, a variant of these methods may then be a functional and plausible model for learning orthogonalization in the cerebellum.

5.3.3 INPUT VARIABLE SPLITTING

In order to self-learn the Cartesian inverse dynamics of the two link manipulator, we had to restrict ourselves to a portion of the manipulator's reach. This was necessary because

certain functional relationships in the inverse kinematics of the two link manipulator could not be represented as a sum of polynomials over the whole of the manipulator's reach. This represents a limitation imposed by the use of K-G polynomials as a basis set.

As mentioned in chapter 3, one could use several unique sum of polynomials representations for each portion of the manipulator's reach. One would thus be modelling the functional relationships over the whole space as piecewise linear, quadratic, or quartic, etc., according to the order of the sum of polynomials representations used in each region. Clearly a more general class of functions can be modelled in this way. It may be possible that one could use a method that we call input variable splitting to achieve the same end.

By input variable splitting, we mean that a single input variable from a control point of view, is encoded on several input variables from the point of view of the sum of polynomials estimator. It is best illustrated by an example.

Consider a sum of polynomials representation for the function,

$$\overset{*}{f} = \sin(2\pi z) \tag{5.58}$$

over the range,

$$-1 < z < 1 \tag{5.59}$$

Clearly a fifth order sum of polynomials representation is required to achieve a minimal likeness of $\overset{*}{f}$. An example of such a representation is,

$$\sin(2\pi z) \simeq 5.7z - 28.4z^3 + 22.7z^5 \tag{5.60}$$

which was obtained by noting that even power terms were not needed and then choosing the coefficients of the odd power terms such that a match is achieved at z equal to 1/4, 1/2, and 1.

Now assume that the information of variable z is split among 4 input variables as follows,

$$z_i = \max\{-1, \min\{1, 4z + 5 - 2i\}\} \qquad (5.61)$$

Note that over most of the range of z, the variables z_i are saturated at either -1 or 1. It is only over specific portions of the space that each z_i varies. It is possible to represent $\sin(2\pi z)$ as a sum of polynomials in z_i, namely,

$$\sin(2\pi z) \simeq - z_1^2 + z_2^2 - z_3^2 + z_4^2 \qquad (5.62)$$

Figure 5.6 shows the variables, z_i, as functions of z. Figure 5.7 shows a plot of $\sin(2\pi z)$ along with the two sum of polynomials representations. It can be seen that the representation using input variable splitting is much better.

It remains to be determined whether Interference Minimization or related learning algorithms can be made to work when input variables are split in this manner. Efforts to develop such methods of learning piecewise non-linear estimates of functions are warranted as it could overcome the limitations we have encountered using estimates based on sums of K-G polynomials. It might make it possible to learn the Cartesian inverse dynamics over the whole of the two link manipulator's reach, and likewise in other situations where functions are not well represented as sums of polynoials over the region of interest.

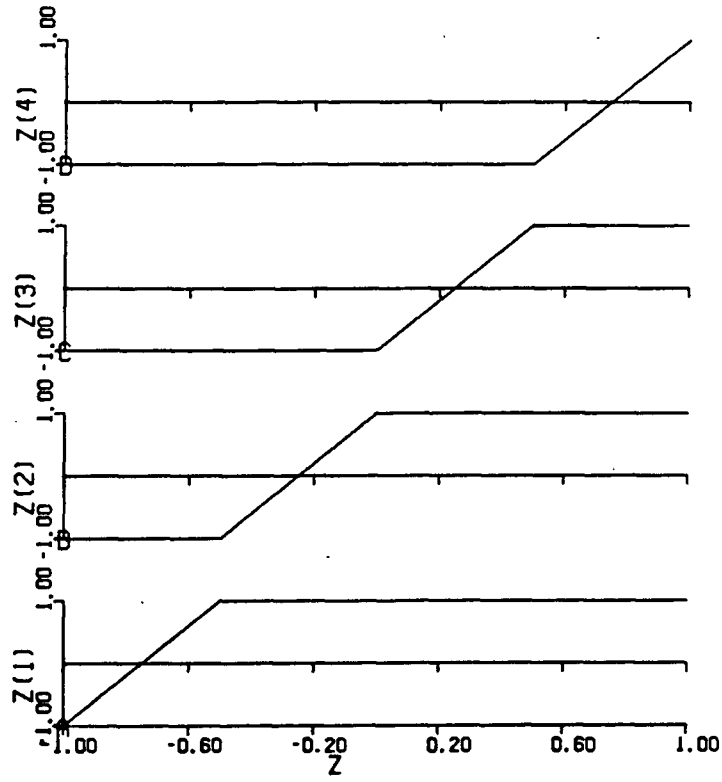


Figure 5.6 Example of input variable splitting

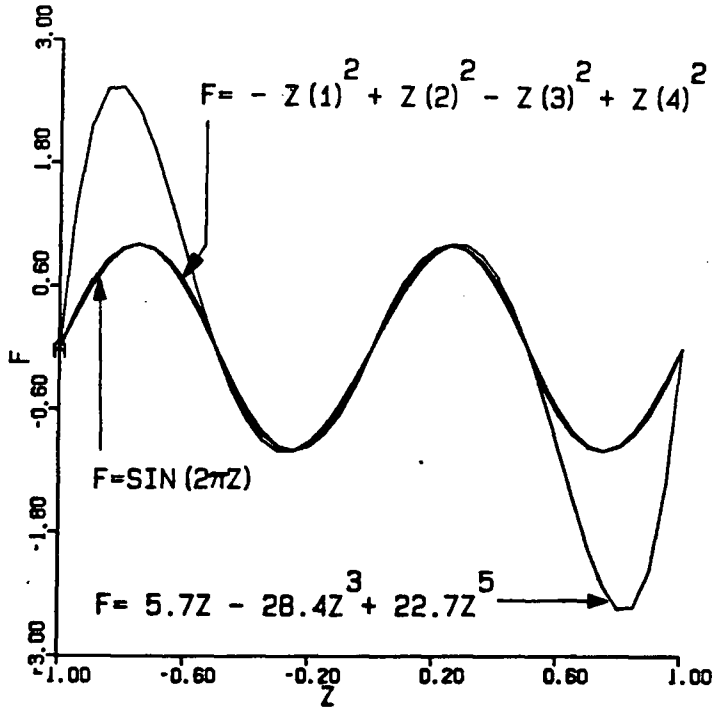


Figure 5.7 Representations of sin(2πz)

With regards to the Cerebellar Model, it is unlikely that input variables are split in the previously described fashion. A more plausible scenario is outlined in the following.

Consider an input variable to the cerebellum from a control point of view. One such variable could be a joint angle, say at the elbow. The position of the joint can be effectively encoded by signals representing the tension or extension in the various muscles acting upon the joint. Since there are numerous muscles and they are generally paired into opponents, it is reasonable to assume that some signals would decrease and others would increase as the joint rotated from its minimum to maximum angular displacement. Also it is reasonable to assume that some signals would change from low levels to saturated high levels over narrow ranges of joint movement while others would change similarly over large ranges of joint movement.

Figure 5.8 shows a simple example where the joint angle, is encoded on several different input variables. If these variables are taken two at a time and multiplied together, the products form functions such as those shown in figure 5.9. Note how these products resemble spline functions. Some are narrow and some are rather wide. Consider now a more realistic physiological scenario in which a parameter such as a joint angle is encoded on the signals coming from perhaps thousands of muscle fiber sensors. If these signals are taken randomly, four or five at a time as they are in the Mossy Fiber - Granule Cell network in the cerebellum, and multiplied together, the set of basis functions that result probably contains many spline-like

functions. This hypothesis remains to be tested, however.

As pointed out by Klett [26], spline functions have desirable properties in terms of minimizing learning interference. Spline functions can be used to form estimates of a more general class of functions than K-G polynomials. Many such spline functions may be required in order to effectively span the domain of the functions to be estimated, however, many such splines could be generated in the cerebellum. Incorporation of spline functions into a modified Klett Cerebellar Model would render the model more similar to the previous model proposed by Albus[1]. The modified Klett Model would retain the advantage of requiring many fewer weighting coefficients than the Albus Model, because of its use of continuous variables versus essentially binary variables in the Albus Model. Use of spline functions would free the Klett Model from its previous disadvantage of only being applicable for estimating functions that are well represented by polynomials; it would gain the generality of the Albus Model in terms of functions that can be learned. Finally, the Albus Model is based on the implausable assumption that each discernable point in its input space is encoded by a set of maximally active input variables. The modified Klett Model, in contrast, would be based on a pattern of sensory data encoding that is more consistent with physiological evidence.

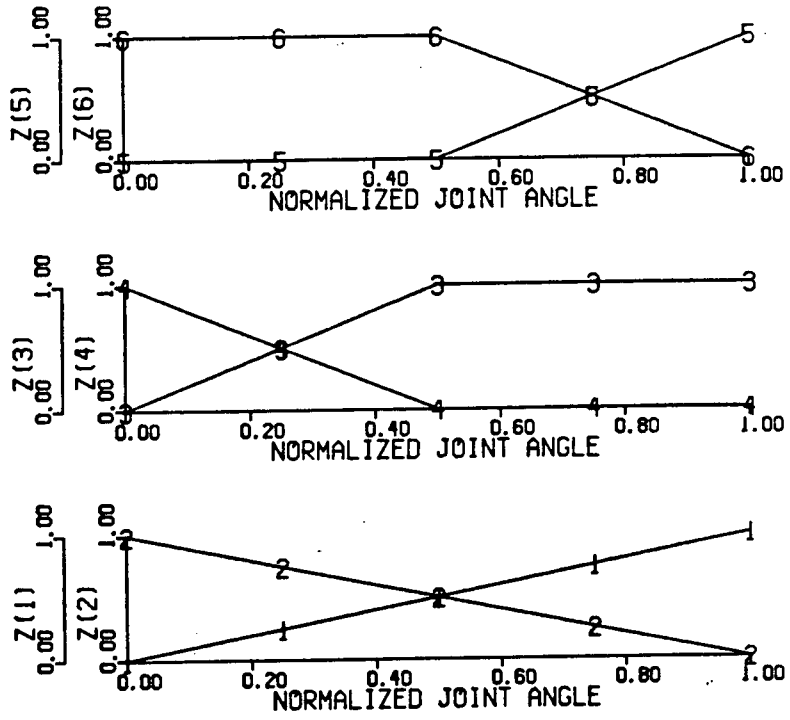


Figure 5.8 Encoding of joint angle by input variable splitting

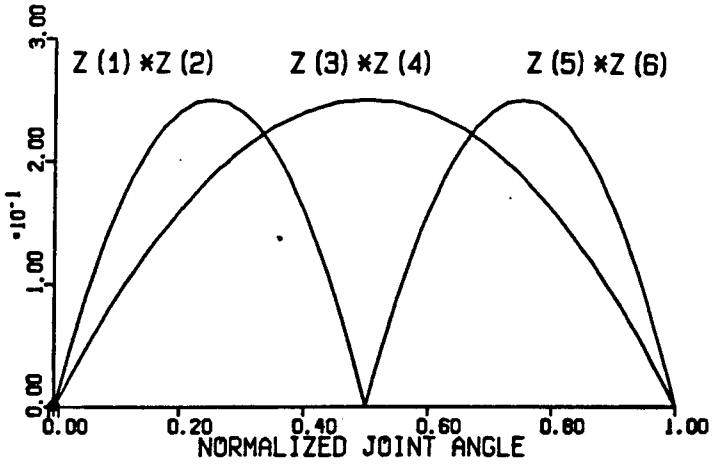


Figure 5.9 Example of products of split input variables representing joint angle

BIBLIOGRAPHY

[1] J.S. Albus, Theoretical and Experimental Aspects of a Cerebellar Model, Ph.D. Thesis, University of Maryland, December 1972.

[2] J.S. Albus, Brains, Behavior & Robotics, BYTE Publications, Peterborough, New Hampshire, 1981.

[3] J.H.J Allum, "A Least Mean Squares CUBIC Algorithm for On-Line Differential of Sampled Analog Signals", IEEE Trans. Comp., C-24, p.585, June 1975.

[4] R.B. Asher, D. Andrisani II & P. Dorato, "Bibliography on Adaptive Control Systems", Proc. IEEE, Vol.64, No.8, p.1226, August 1976.

[5] P.R. Belanger, "Comments on `A Learning Method for System Identification'", IEEE Trans. Auto. Cont., AC-13, p.207, April 1968.

[6] V.J. Bidwell, "Regression Analysis of Nonlinear Catchment Systems", Water Resources Research, Vol.7, No.5, p.1118, October 1971.

[7] S.A. Billings, "Identification of Nonlinear Systems - A Survey", IEE Proc. D, Vol.127, p.272, November 1980.

[8] R.R. Bitmead & B.D.O. Anderson, "Performance of Adaptive Estimation Algorithms in Dependent Random Environments", IEEE Trans. Auto. Cont., AC-25, p.788, August 1980.

[9] R.E. Butler & E.V. Bohn, "An Automatic Identification Technique for a Class of Nonlinear Systems", IEEE Trans. Auto. Cont., AC-11, p.292, April 1966.

[10] W.C. Chan & C.C. Babu, "An Algorithm for Function Restoration Using Eigenvectors", Int. J. Cont., Vol.14, No.4, p.673, 1971.

[11] W.C. Chan & C.C. Babu, "Function Restoration Using Linear Programming", Int. J. Cont., Vol.14, No.4, p.681, 1971.

[12] L.A. Crum & S.H. Hu, "Convergence of the Quantizing Learning Method for System Identification", IEEE Trans. Auto. Cont., AC-13, p.297, June 1968.

[13] J.C. Eccles, "The Cerebellum as a Computer: Patterns in Space and Time", J. Physiol. Lond., 229, No.1, p.1, 1973.

[14] E. Eweda & M. Odile, "Convergence of an Adaptive Linear Estimation Algorithm", IEEE Trans. Auto. Cont., AC-29, p.119, February 1984.

[15] P. Eykhoff, System Identification, John Wiley & Sons, New York, 1974.

[16] K.S. Fu, "Learning Control Systems - Review and Outlook", IEEE Trans. Auto. Cont., AC-15, p.210, April 1970.

[17] K.S. Fu, "Learning Control Systems and Intelligent Control Systems: An Intersection of Artificial Intelligence and Automatic Control", IEEE Trans. Auto. Cont., AC-16, p.70, February 1971.

[18] D. Gabor, W.P.L. Wilby & R. Woodcock, "A Universal Non-Linear Filter, Predictor and Simulator Which Optimizes Itself by a Learning Process", IEE Proc. D, p.422, July 1960.

[19] B.R. Gaines, "Stochastic Computing Systems" in Advances in Information Science, J.T. Tou, ed., Plenum Press, New York, 1969.

[20] S.S. Godbole & C.F. Smith, "A New Approach Using the Inverse Sytem", IEEE Trans, Auto. Cont., AC-17, p.698, October 1972.

[21] L.R. Goke & K.L. Doty, "Design of a Random-Pulse Computer for Classifying Binary Patterns", IEEE Trans. Comp., C-21, p.1347, December 1972.

[22] T.C. Hsia & V. Vimolvanich, "An On-Line Technique for System Identification", IEEE Trans. Auto. Cont., AC-14, p.92, February 1969.

[23] C.K. Huscroft & P.D. Lawrence, "Randomization of Pulse-Rate-Encoded Signals for Use in Stochastic Computers", Electron. Lett., Vol.20, No.12, 7th June 1984.

[24] A.G. Ivakhnenko, "Polynomial Theory of Complex Systems", IEEE Trans. Sys. Man Cyber., SMC-1, p.364, October 1971.

[25] C.R. Johnson, "An Implementation of the Multidimensional Modified LMS Algorithm", IEEE Trans. Aero. Elect., AES-16, p.398, May 1980.

[26] D. Klett, A Cerebellum-Like Learning Machine, M.A.Sc. Thesis, University of British Columbia, July 1979.

[27] A.J. Koivo & Ten-Huei Guo, "Adaptive Linear Controller for Robotic Manipulars", IEEE Trans. Auto. Cont., AC-28, p.162, February 1983.

[28] C.S.G. Lee, "Robot Arm Kinematics, Dynamics, and Control", Computer, Vol.15, No.12, p.62, December 1982.

[29] Ching-Fang Lin, "Advanced Controller Design for Robot Arms", IEEE Trans. Auto. Cont., AC-29, p.350, April 1984.

[30] J.Y.S. Luh, M.W. Walker & R.C. Paul, "Resolved-Acceleration Control of Mechanical Manipulators", IEEE Trans. Auto. Cont, AC-25, p.468, June 1980.

[31] D. Marr, "A Theory of the Cerebellar Cortex", J. Physiol. Lond., 202, p.437, 1969.

[32] P. Mars & H.R. McLean, "Implementation of Linear Programming with a Digital Stochastic Computer", Electron. Lett., Vol.12, No.20, p.516, 30th September 1976.

[33] P. Mars & H.R. McLean, "High-Speed Matrix Inversion by Stochastic Computer", Electron. Lett., Vol.12, No.18, p.457, 2nd September 1976.

[34] J.M. Martin-Sanchez, "A New Solution to Adaptive Control", Proc. IEEE, Vol.64, No.8, p.1209, August 1976.

[35] J.M. Mendel, Discrete Techniques of Parameter Estimation, Marcel Dekker, Inc., New York, 1973.

[36] J.M. Mendel, "Gradient Estimation Algorithms for Equation Error Formulations", IEEE Trans. Auto. Cont., AC-19, p.820, December 1974.

[37] E.N. Moore, Theoretical Mechanics, John Wiley & Sons, New York, 1983.

[38] V.B. Mountcastle, ed., Medical Physiology, 13th edition, C.V. Mosby Company, Saint Louis, 1974.

[39] J.I. Nagumo & A. Noda, "A Learning Method for System Identification", IEEE Trans. Auto. Cont., AC-12, p.282, June 1967.

[40] K.S. Narendra & M.A.L. Thathachar, "Learning Automata - A Survey", IEEE Trans. Sys. Man Cyber., SMC-4, p.323, July 1974.

[41] Z.J. Nikolic & K.S. Fu, "An Algorithm for Learning Without External Supervision and Its Application to Learning Control Systems", IEEE Trans. Auto. Cont., AC-11, p.414, July 1966.

[42] N.J. Nilsson, "Learning Machines", McGraw-Hill, New York, 1965.

[43] B. Noble & J.W. Daniels, Applied Linear Algebra, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1977.

[44] R.C. Paul, B. Shimano & G.E. Mayer, "Kinematic Control Equations for Simple Manipulators", IEEE Trans. Sys. Man Cyber., SMC-11, p.449, June 1981.

[45] R.C. Paul, B. Shimano & G.E. Mayer, "Differential Kinematic Control Equations for Simple Manipulators", IEEE Trans. Sys. Man Cyber., SMC-11, p.456, June 1981.

[46] R.C. Paul, Robot Manipulators: Mathematics, Programming and Control, MIT Press, Cambridge, Mass., 1981.

[47] W.J. Poppelbaum, C. Afuso & J.W. Esch, "Stochastic Computing Elements and Systems" in Proceedings AFIPS, Fall Joint Computer Conference, Vol.31, p.635, Books, Inc., New York, 1967.

[48] S.T. Ribeiro, "Random-Pulse Machines", IEEE Trans. Comp., EC-16, p.261, June 1967.

[49] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", Psycol. Rev., Vol.65, No.6, p.386, 1958.

[50] R.J. Roy & J. Sherman, "A Learning Technique for Volterra Series Representation", IEEE Trans. Auto. Cont., AC-12, p.761, December 1967.

[51] G.N. Saridis, "Learning Applied to Successive Approximation Algorithms", IEEE Trans. Sys. Sci. Cyber., SSC-6, p.97, April 1970.

[52] G.N. Saradis, "Toward the Realization of Intelligent Controls", Proc. IEEE, Vol.67, p.1115, August 1979.

[53] G.N. Saridis, "Application of Pattern Recognition Methods to Control Systems", IEEE Trans. Auto. Cont., AC-26, p.638, June 1981.

[54] J. Sklansky, "Learning Systems for Automatic Control", IEEE Trans. Auto. Cont., AC-11, p.6, January 1966.

[55] D.E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses", IEEE Trans. Man Mach. Sys., MMS-10, p.47, June 1969.

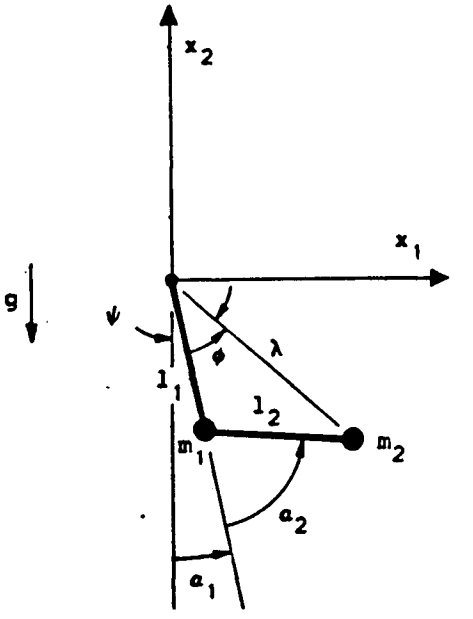
[56] B. Widrow, N.K. Gupta & S. Maitra, "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems", IEEE Trans. Sys. Man Cyber., SMC-3, p.455, September 1973.

[57] B. Widrow, J.M. McCool, M.G. Larimore & C.R. Johnson Jr., "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter", Proc. IEEE, Vol.64, p.1151, August 1976.

[58] ACSL User Guide/Reference Manual, Mitchell and Gauthier, Assoc., Inc., Concord, Mass.

APPENDIX A

ANALYTICAL KINEMATICS OF THE TWO LINK MANIPULATOR



THE TWO LINK MANIPULATOR

THE DIRECT KINEMATICS

From the figure the direct position kinematics can be immediately obtained,

$$x_1 = l_1\sin(a_1) + l_2\sin(a_1+a_2)$$

$$x_2 = -l_1\cos(a_1) - l_2\cos(a_1+a_2)$$

Differentiation of these relationships yields the direct velocity kinematics,

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}l_1\cos(a_1) + l_2\cos(a_1+a_2) & , & l_2\cos(a_1+a_2)\\ l_1\sin(a_1) + l_2\sin(a_1+a_2) & , & l_2\sin(a_1+a_2)\end{bmatrix} \begin{bmatrix}\dot{a}_1\\ \dot{a}_2\end{bmatrix}$$

which can be rearranged as,

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}l_1\cos(a_1) & , & l_2\cos(a_1+a_2)\\ l_1\sin(a_1) & , & l_2\sin(a_1+a_2)\end{bmatrix} \begin{bmatrix}\dot{a}_1\\ \dot{a}_1+\dot{a}_2\end{bmatrix}$$

and represented as,

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \mathbf{M}_1 \begin{bmatrix}\dot{a}_1\\ \dot{a}_1+\dot{a}_2\end{bmatrix}$$

The direct velocity kinematics can be differentiated to yield the direct acceleration kinematics,

$$\begin{bmatrix}\ddot{x}_1\\ \ddot{x}_2\end{bmatrix} = \begin{bmatrix}l_1\cos(a_1) & , & l_2\cos(a_1+a_2)\\ l_1\sin(a_1) & , & l_2\sin(a_1+a_2)\end{bmatrix} \begin{bmatrix}\ddot{a}_1\\ \ddot{a}_1+\ddot{a}_2\end{bmatrix}$$

$$+ \begin{bmatrix}-l_1\sin(a_1) & , & -l_2\sin(a_1+a_2)\\ l_1\cos(a_1) & , & l_2\cos(a_1+a_2)\end{bmatrix} \begin{bmatrix}\dot{a}_1^{\,2}\\ (\dot{a}_1+\dot{a}_2)^2\end{bmatrix}$$

which can be represented as,

$$\begin{bmatrix}\ddot{x}_1\\ \ddot{x}_2\end{bmatrix} = \mathbf{M}_1 \begin{bmatrix}\ddot{a}_1\\ \ddot{a}_1+\ddot{a}_2\end{bmatrix} + \mathbf{M}_2 \begin{bmatrix}\dot{a}_1^{\,2}\\ (\dot{a}_1+\dot{a}_2)2\end{bmatrix}$$

THE INVERSE KINEMATICS

Referring again to the figure, the cosine rule can be applied to yield,

$$\cos(\pi-a_2) = (\lambda^2-l_1^{\,2}-l_2^{\,2})/(-l_1l_2)$$

and thus,

$$\cos(a_2) = (\lambda^2-l_1^{\,2}-l_2^{\,2})/l_1l_2$$

By the rule of Pythagoras we have that,

$$\lambda^2 = x_1^{\,2} + x_2^{\,2}$$

and thus,

$$\cos(a_2) = (x_1^{\,2}+x_2^{\,2}-l_1^{\,2}-l_2^{\,2})/l_1l_2$$

or equivalently,

$$a_2 = \arccos((x_1^{\,2}+x_2^{\,2}-l_1^{\,2}-l_2^{\,2})/l_1l_2)$$

which is the inverse position kinematics function for a_2.

The inverse position kinematics function for a_1 can be

obtained by noting that,

$$\tan(\phi) = (l_2\sin(a_2))/(l_1+l_2\cos(a_2))$$

and,

$$\tan(\psi) = x_1/(-x_2)$$

Since,

$$a_1 = \psi - \phi$$

this means that,

$$a_1 = \arctan(x_1,-x_2) - \arctan(l_2\sin(a_2),l_1+l_2\cos(a_2))$$

Note that the arctan function used here has two arguments and is thus assumed to yield a result in the range $-\pi$ to π.

An alternative representation of the inverse position kinematics for a_1 can be obtained by noting that,

$$\cos(a_1) = \cos(\psi-\phi) = \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\phi)$$

By inspection of the figure we have,

$$\cos(\psi) = -x_2/\lambda$$

and,

$$\sin(\psi) = x_1/\lambda$$

Application of the cosine rule yields,

$$\cos(\phi) = (\lambda^2+l_1^2-l_2^2)/2l_1\lambda$$

Application of the sine rule yields,

$$\sin(\phi)/l_2 = \sin(\pi-a_2)/\lambda = \sin(a_2)/\lambda$$

and hence,

$$\sin(\phi) = l_2\sin(a_2)/\lambda$$

Substituting these expressions into our previous expression for $\cos(a_1)$ yields,

$$\cos(a_1) = -x_2(\lambda^2+l_1^2-l_2^2)/2l_1\lambda^2 + x_1l_2\sin(a_2)/\lambda^2$$

Subsitituting in the previous expression for λ^2 and combining

terms that are equal to $\cos(a_2)$ yields,

$$\cos(a_1) = [x_1 l_1 \sin(a_2) - x_2(l_1+l_2\cos(a_2))]/(x_1^2+x_2^2)$$

another form of the inverse kinematics function of a_1.

From the direct velocity kinematics we have,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \mathbf{M}_1 \begin{bmatrix} \dot{a}_1 \\ \dot{a}_1+\dot{a}_2 \end{bmatrix}$$

which can be rearranged to yield the inverse velocity kinematics as,

$$\begin{bmatrix} \dot{a}_1 \\ \dot{a}_1+\dot{a}_2 \end{bmatrix} = \mathbf{M}_1^{-1} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}$$

or equivalently,

$$\begin{bmatrix} \dot{a}_1 \\ \dot{a}_1+\dot{a}_2 \end{bmatrix} = \frac{1}{l_1 l_2 \sin(a_2)} \begin{bmatrix} l_2\sin(a_1+a_2) & -l_2\cos(a_1+a_2) \\ -l_1\sin(a_1) & l_1\cos(a_1) \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}$$

From the direct acceleration kinematics we have,

$$\begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} = \mathbf{M}_1 \begin{bmatrix} \ddot{a}_1 \\ \ddot{a}_1+\ddot{a}_2 \end{bmatrix} + \mathbf{M}_2 \begin{bmatrix} \dot{a}_1^2 \\ (\dot{a}_1+\dot{a}_2)2 \end{bmatrix}$$

which can be rearranged to yield the inverse acceleration kinematics as,

$$\begin{bmatrix} \ddot{a}_1 \\ \ddot{a}_1+\ddot{a}_2 \end{bmatrix} = \mathbf{M}_1^{-1} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} - \mathbf{M}_1^{-1}\mathbf{M}_2 \begin{bmatrix} \dot{a}_1^2 \\ (\dot{a}_1+\dot{a}_2)2 \end{bmatrix}$$

or equivalently,

$$\begin{bmatrix} \ddot{a}_1 \\ \ddot{a}_1+\ddot{a}_2 \end{bmatrix} = \frac{1}{l_1 l_2 \sin(a_2)} \begin{bmatrix} l_2\sin(a_1+a_2) & -l_2\cos(a_1+a_2) \\ -l_1\sin(a_1) & l_1\cos(a_1) \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} - \frac{1}{l_1 l_2 \sin(a_2)} \begin{bmatrix} -l_1 l_2\cos(a_2) & -l_1^2 \\ l_1^2 & l_1 l_2\cos(a_2) \end{bmatrix} \begin{bmatrix} \dot{a}_1^2 \\ (\dot{a}_1+\dot{a}_2)^2 \end{bmatrix}$$

APPENDIX B

ANALYTICAL DYNAMICS OF THE TWO LINK MANIPULATOR

INVERSE DYNAMICS

The Lagrangian [46,37] is defined as the difference between the kinetic energy of a system, K, and the potential energy of a system, P,

$$L = K - P$$

The inverse dynamics equations are obtained as,

$$\tau_i = \frac{\delta}{\delta t}\left[\frac{\partial L}{\partial \dot{a}_i}\right] - \frac{\partial L}{\partial a_i}$$

The derivation of the inverse dynamics equations begins by noting that the kinetic energy of mass m_1 is,

$$K_1 = 1/2\ m_1 v_1^2 = 1/2\ m_1 l_1^2 \dot{a}_1^2$$

which is equal to 0 when not moving. The potential energy of mass m_2 is,

$$P_1 = -m_1 gh = -m_1 g l_1 \cos(a_1)$$

which is equal to 0 when $x_2=0$, ie. at the height of the origin. The kinetic energy of mass m_2 is,

$$K_2 = 1/2\ m_2 v_2^2$$

The square of the velocity of mass m_2 can be obtained from the direct velocity kinematics equations given in appendix A by noting that,

$$v_2^2 = \dot{x}_1^2 + \dot{x}_2^2$$

and hence,

$$v_2^2 = l_1^2 \dot{a}_1^2 + l_2^2(\dot{a}_1^2 + 2\dot{a}_1\dot{a}_2 + \dot{a}_2^2) + 2l_1 l_2 \dot{a}_1(\dot{a}_1 + \dot{a}_2)\cos(a_2)$$

Thus the kinetic energy of mass m_2 is,

$$K_2 = 1/2\ m_2 l_1^2 \dot{a}_1^2 + 1/2\ m_2 l_2^2 (\dot{a}_1^2 + 2\dot{a}_1\dot{a}_2 + \dot{a}_2^2) + m_2 l_1 l_2 \cos(a_2)(\dot{a}_1^2 + \dot{a}_1\dot{a}_2)$$

The potential energy of mass m_2 is,

$$P_2 = m_2 g h = -m_2 g l_1 \cos(a_1) - m_2 g l_2 \cos(a_1 + a_2)$$

Combining terms yields the Lagrangian of the two link manipulator,

$$L = 1/2\ (m_1+m_2) l_1^2 \dot{a}_1^2 + 1/2\ m_2 l_2^2 (\dot{a}_1^2 + 2\dot{a}_1\dot{a}_2 + \dot{a}_2^2) + m_2 l_1 l_2 \cos(a_2)(\dot{a}_1^2 + \dot{a}_1\dot{a}_2) + (m_1+m_2) g l_1 \cos(a_1) + m_2 g l_2 \cos(a_1 + a_2)$$

Various derivatives of the Lagrangian need to be obtained and combined to yield the inverse dynamics function for torque τ_1. The derivatives are,

$$\partial L/\partial \dot{a}_1 = (m_1+m_2) l_1^2 \dot{a}_1 + m_2 l_2^2 \dot{a}_1 + m_2 l_2^2 \dot{a}_2 + 2 m_2 l_1 l_2 \cos(a_2)\dot{a}_1 + m_2 l_1 l_2 \cos(a_2)\dot{a}_2$$

$$\delta/\delta t(\partial L/\partial \dot{a}_1) = [(m_1+m_2) l_1^2 + m_2 l_2^2 + 2 m_2 l_1 l_2 \cos(a_2)]\ddot{a}_1 + [m_2 l_2^2 + m_2 l_1 l_2 \cos(a_2)]\ddot{a}_2 - 2 m_2 l_1 l_2 \sin(a_2)\dot{a}_1\dot{a}_2 - m_2 l_1 l_2 \sin(a_2)\dot{a}_2^2$$

$$\partial L/\partial a_1 = -(m_1+m_2) g l_1 \sin(a_1) - m_2 g l_2 \sin(a_1 + a_2)$$

Combining these derivatives yields the inverse dynamics equation for torque τ_1,

$$\tau_1 = d_{11}\ddot{a}_1 + d_{12}\ddot{a}_2 + d_{111}\dot{a}_1^2 + d_{122}\dot{a}_2^2 + d_{112}\dot{a}_1\dot{a}_2 + d_{121}\dot{a}_2\dot{a}_1 + d_1$$

where,

$$d_{11} = (m_1+m_2) l_1^2 + m_2 l_2^2 + 2 m_2 l_1 l_2 \cos(a_2)$$

$$d_{12} = m_2 l_2^2 + m_2 l_1 l_2 \cos(a_2)$$

$$d_{111} = 0$$

$$d_{122} = -m_2 l_1 l_2 \sin(a_2)$$

$$d_{112} = -m_2 l_1 l_2 \sin(a_2)$$

$$d_{121} = -m_2 l_1 l_2 \sin(a_2)$$

$$d_1 = (m_1+m_2) g l_1 \sin(a_1) + m_2 g l_2 \sin(a_1+a_2)$$

Similarly, various derivatives of the Lagrangian need to be obtained and combined to yield the inverse dynamics function for torque τ_2. The derivatives are,

$$\partial L/\partial \dot{a}_2 = m_2 l_2^2 \dot{a}_1 + m_2 l_2^2 \dot{a}_2 + m_2 l_1 l_2 \cos(a_2) \dot{a}_1$$

$$\delta/\delta t(\partial L/\partial \dot{a}_2) = [m_2 l_2^2 + m_2 l_1 l_2 \cos(a_2)] \ddot{a}_1 + m_2 l_2^2 \ddot{a}_2 - m_2 l_1 l_2 \sin(a_2) \dot{a}_1 \dot{a}_2$$

$$\partial L/\partial a_2 = - m_2 l_1 l_2 \sin(a_2) \dot{a}_1^2 - m_2 l_1 l_2 \sin(a_2) \dot{a}_1 \dot{a}_2 - m_2 g l_2 \sin(a_1+a_2)$$

Combining these derivatives yields the inverse dynamics equation for torque τ_2,

$$\tau_2 = d_{21} \ddot{a}_1 + d_{22} \ddot{a}_2 + d_{211} \dot{a}_1^2 + d_{222} \dot{a}_2^2 + d_{212} \dot{a}_1 \dot{a}_2 + d_{221} \dot{a}_2 \dot{a}_1 + d_2$$

where,

$$d_{21} = m_2 l_2^2 + m_2 l_1 l_2 \cos(a_2)$$

$$d_{22} = m_2 l_2^2$$

$$d_{211} = m_2 l_1 l_2 \sin(a_2)$$

$$d_{222} = 0$$

$$d_{212} = 0$$

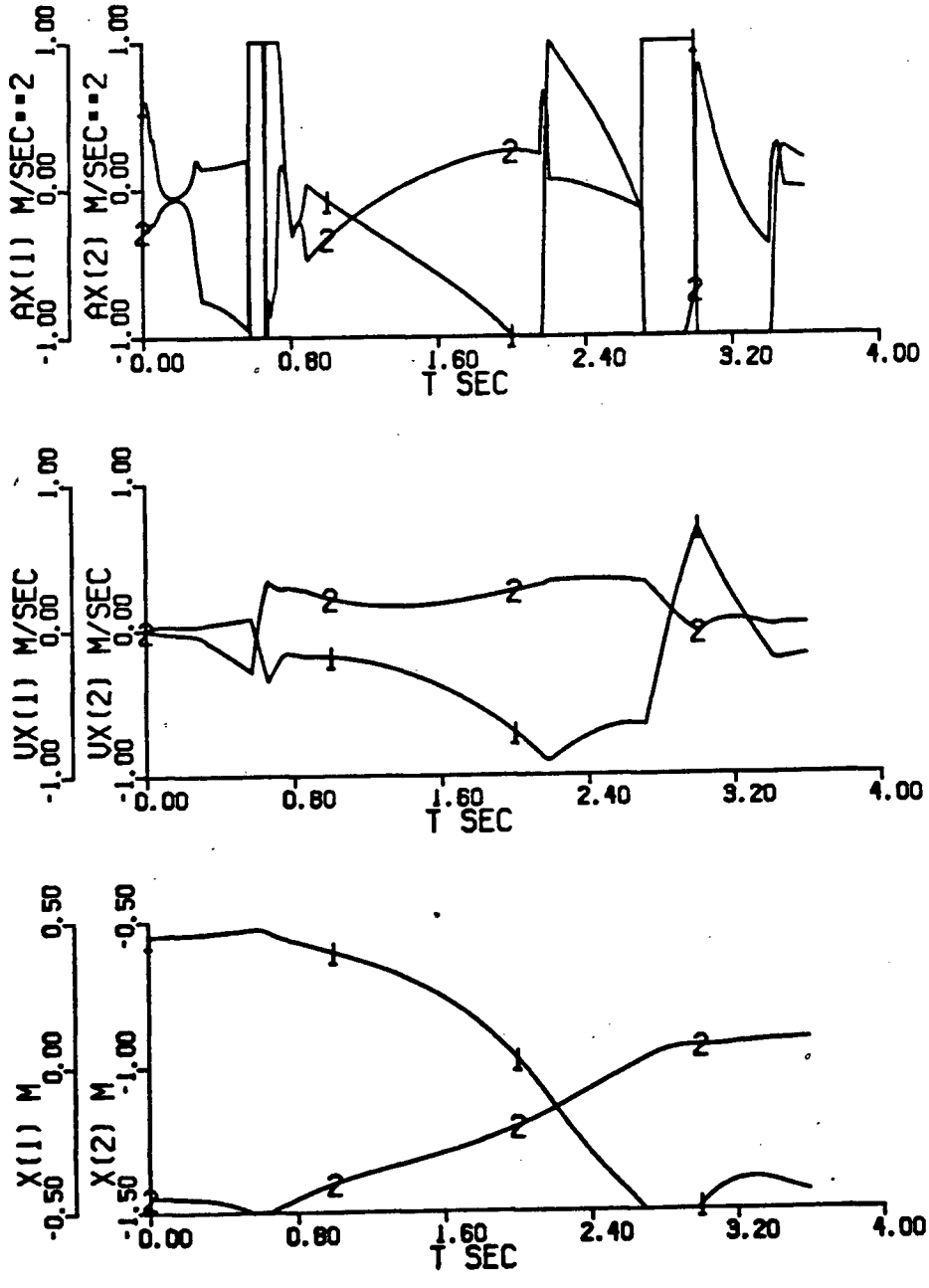
$$d_{221} = 0$$

$$d_2 = m_2 g l_2 \sin(a_1+a_2)$$

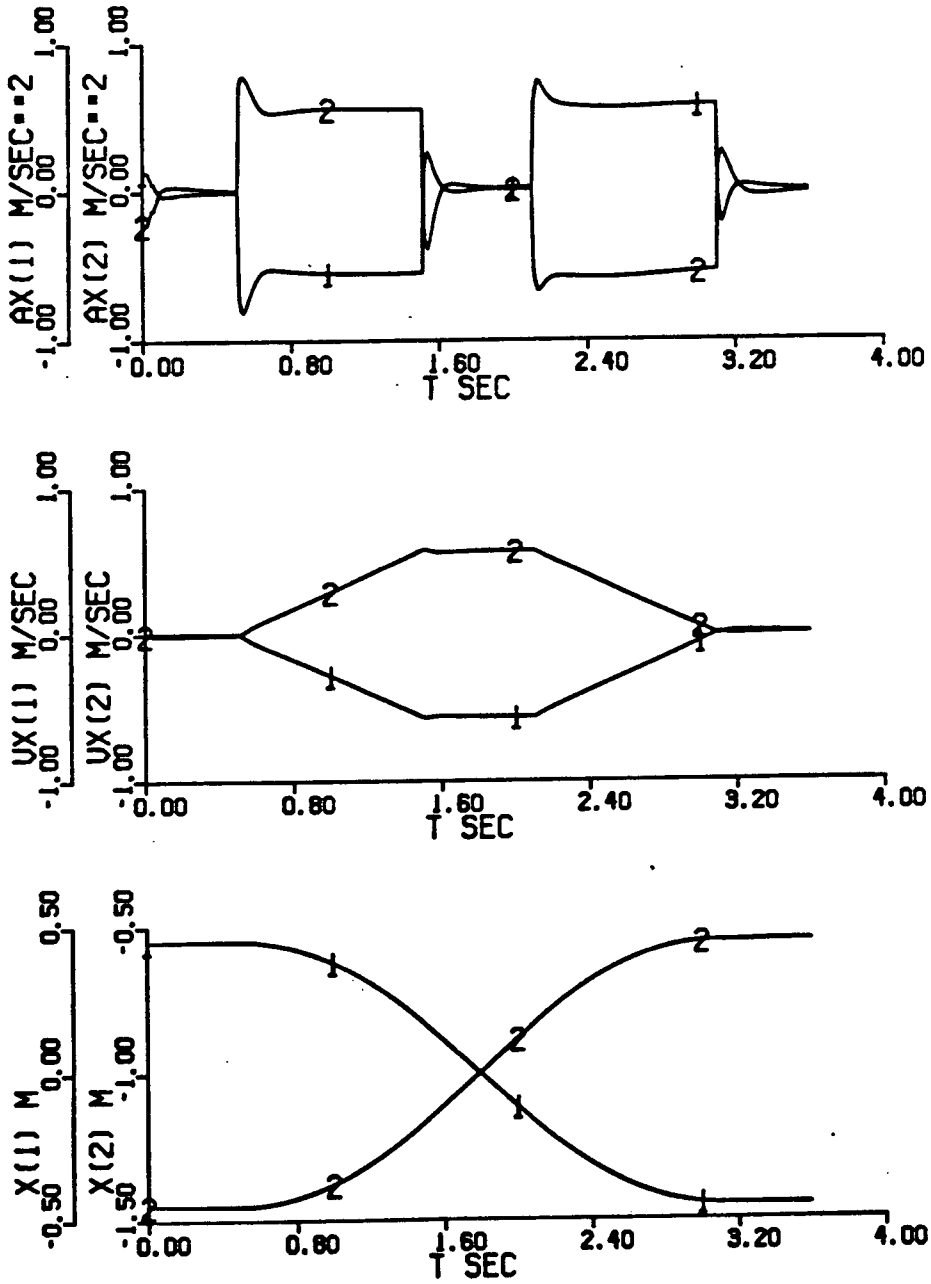
This derivation of the inverse dynamics follows the approach used by Paul [46]. Note, however, that the results differ slightly from those given by Paul as there is an arithmetic error in his derivation.

APPENDIX C

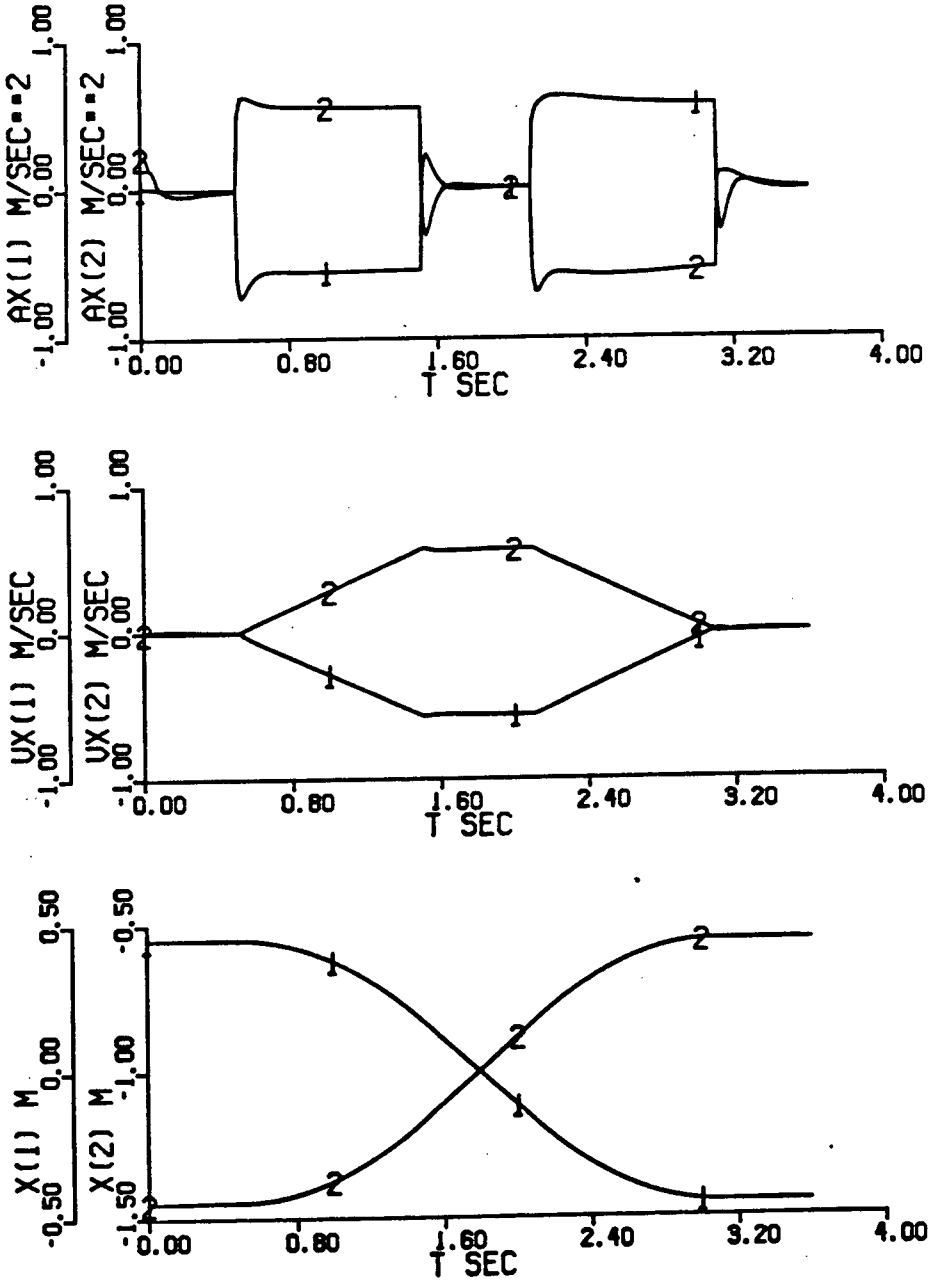
CLOSED LOOP CONTROL OF LINE 1
USING THE SELF-LEARNED CARTESIAN INVERSE DYNAMICS
AFTER 200, 400, 600, 800 AND 1000
TRAINING PATHS



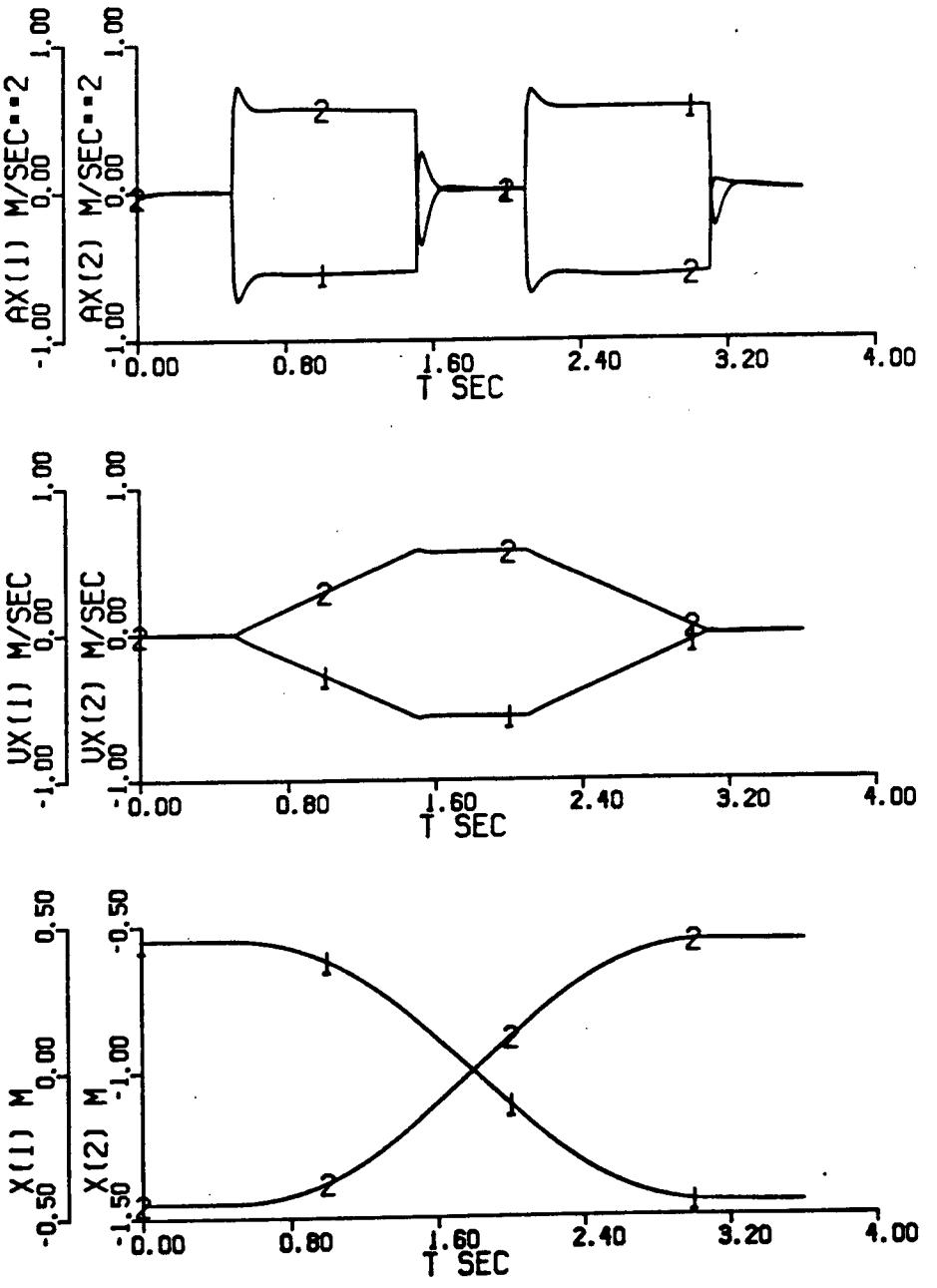
Closed loop control of line 1 using the self-learned Cartesian inverse dynamics after 200 training paths



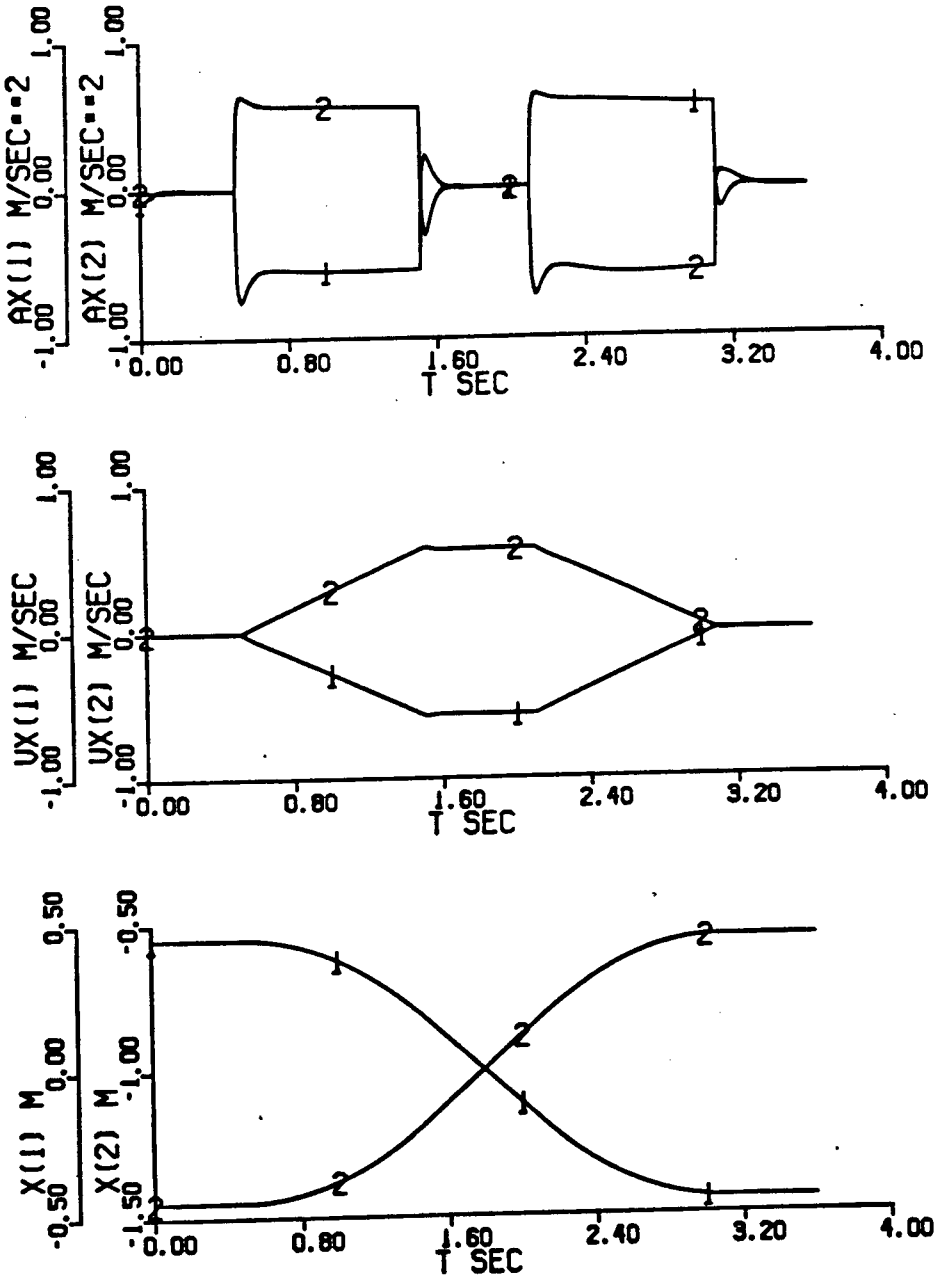
Closed loop control of line 1 using the self-learned Cartesian inverse dynamics after 400 training paths



Closed loop control of line 1 using the self-learned Cartesian inverse dynamics after 600 training paths



Closed loop control of line 1 using the self-learned Cartesian inverse dynamics after 800 training paths



Closed loop control of line 1 using the self-learned Cartesian inverse dynamics after 1000 training paths

APPENDIX D

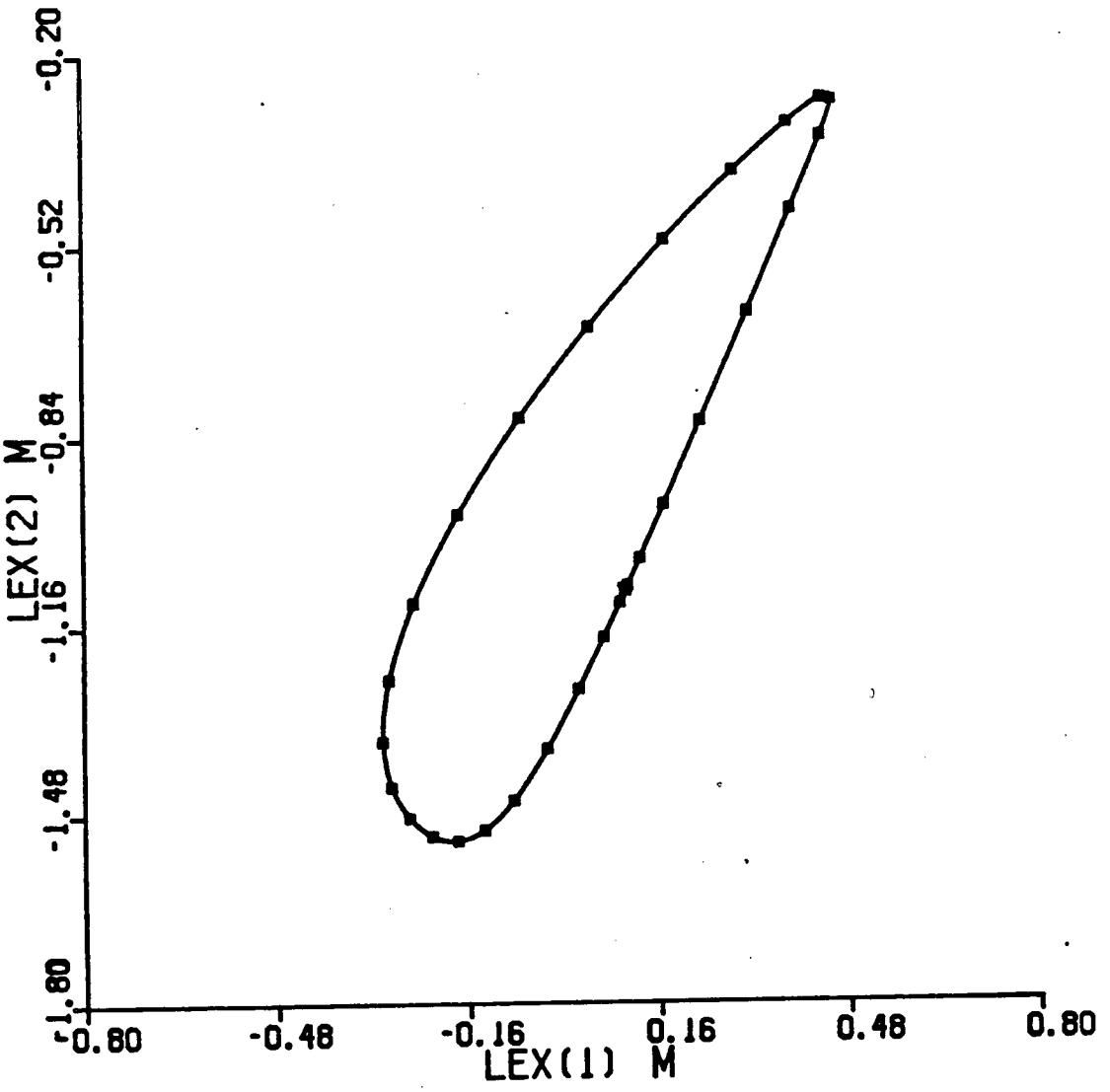
VIEW OF CIRCLE 1

USING THE SELF-LEARNED DIRECT POSITION KINEMATICS

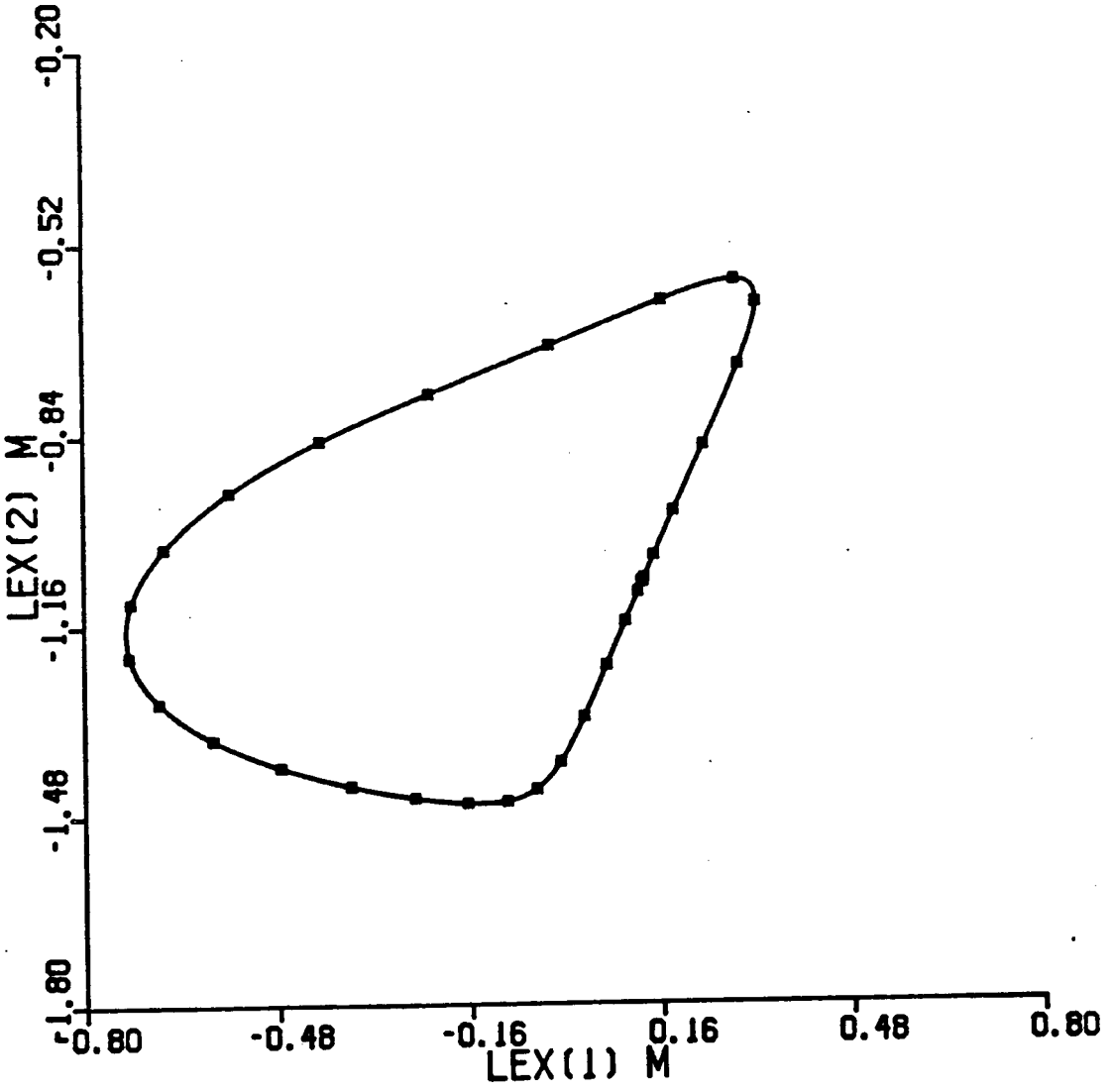
AFTER 200, 400, 600, 800, 1000, 1400, 1800,

2200, 2600, 3000, 3400, 3800, 4200, AND 4600

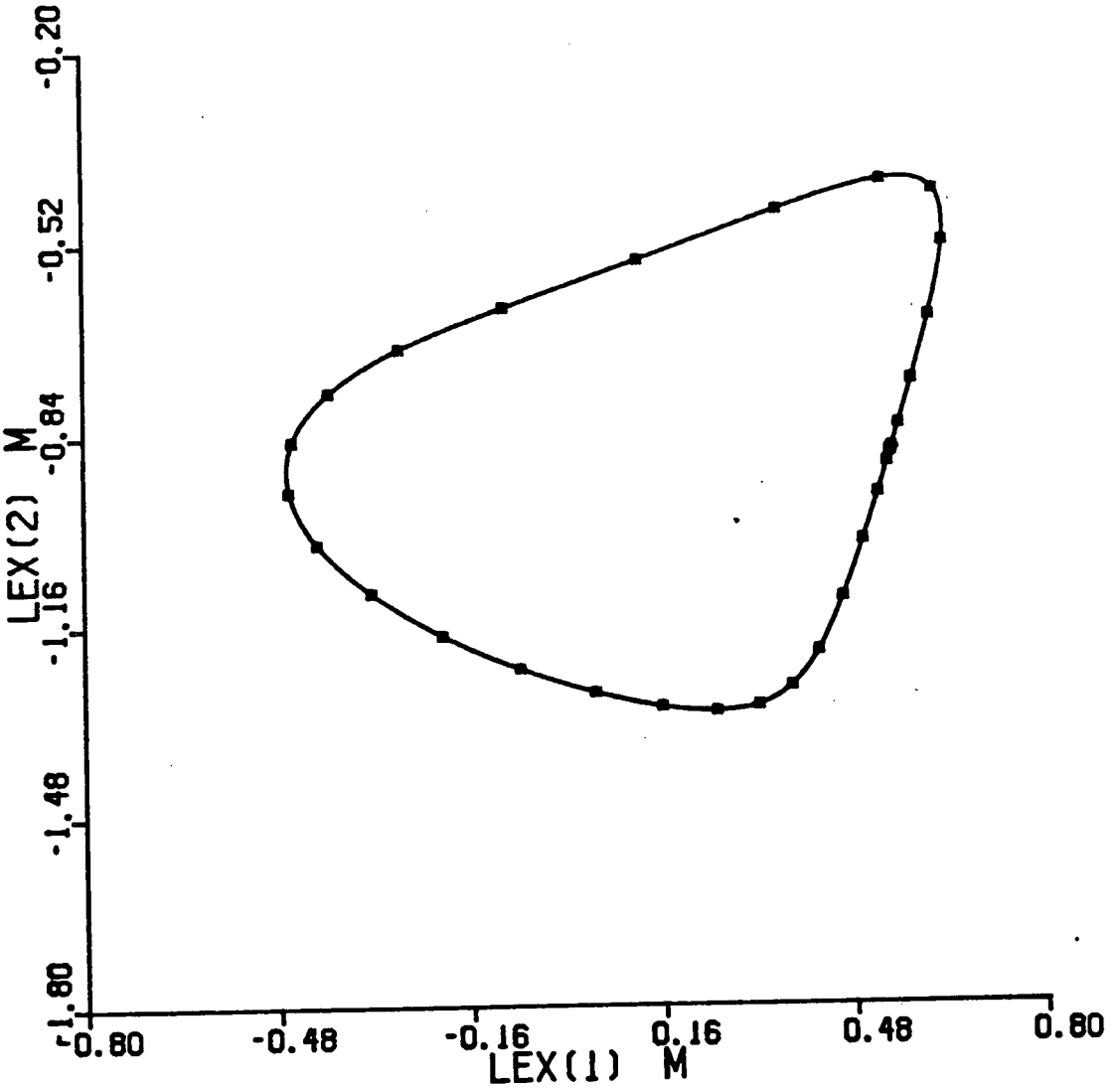
TRAINING PATHS



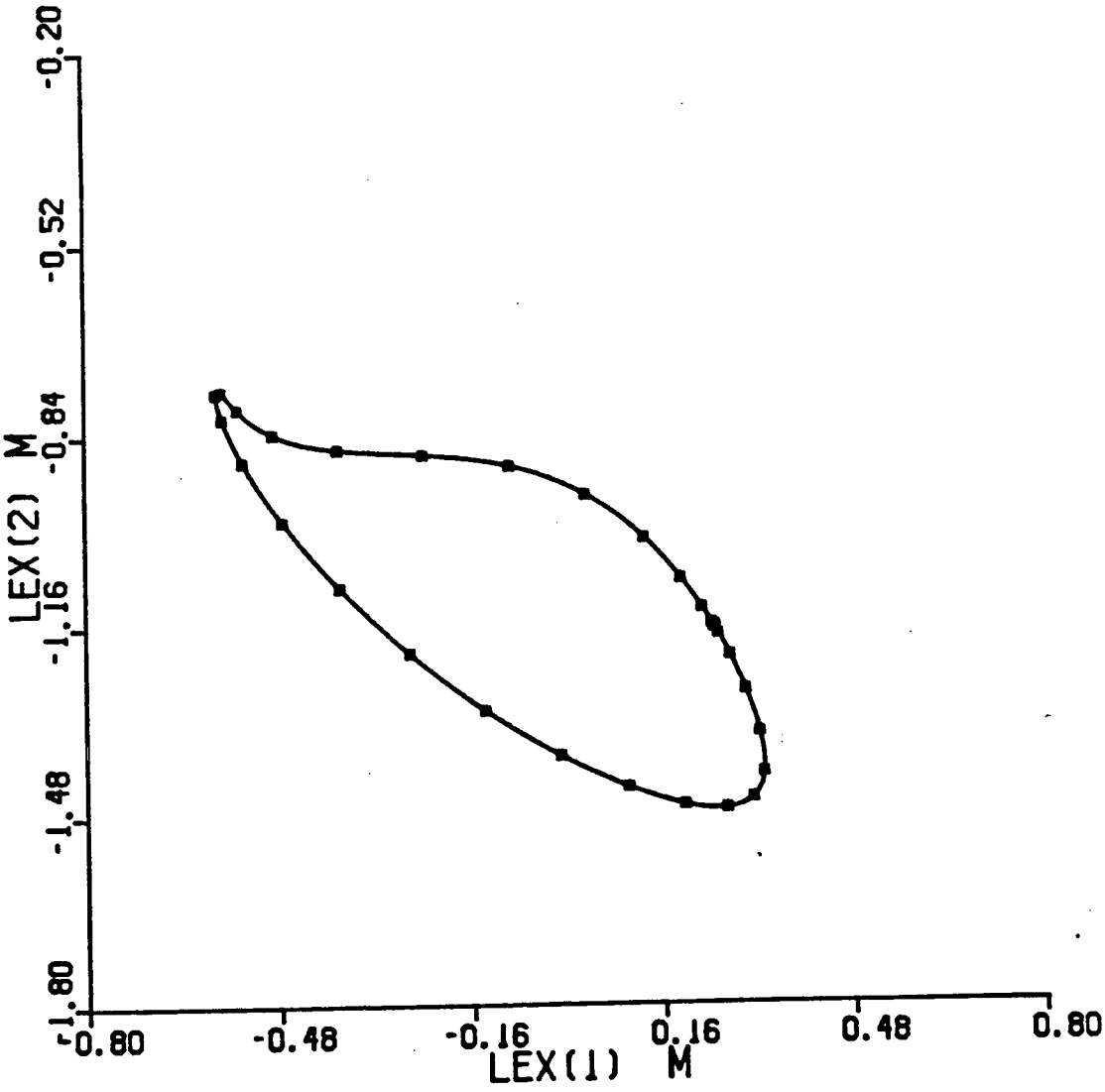
View of circle 1 using the self-learned direct position kinematics after 200 training paths



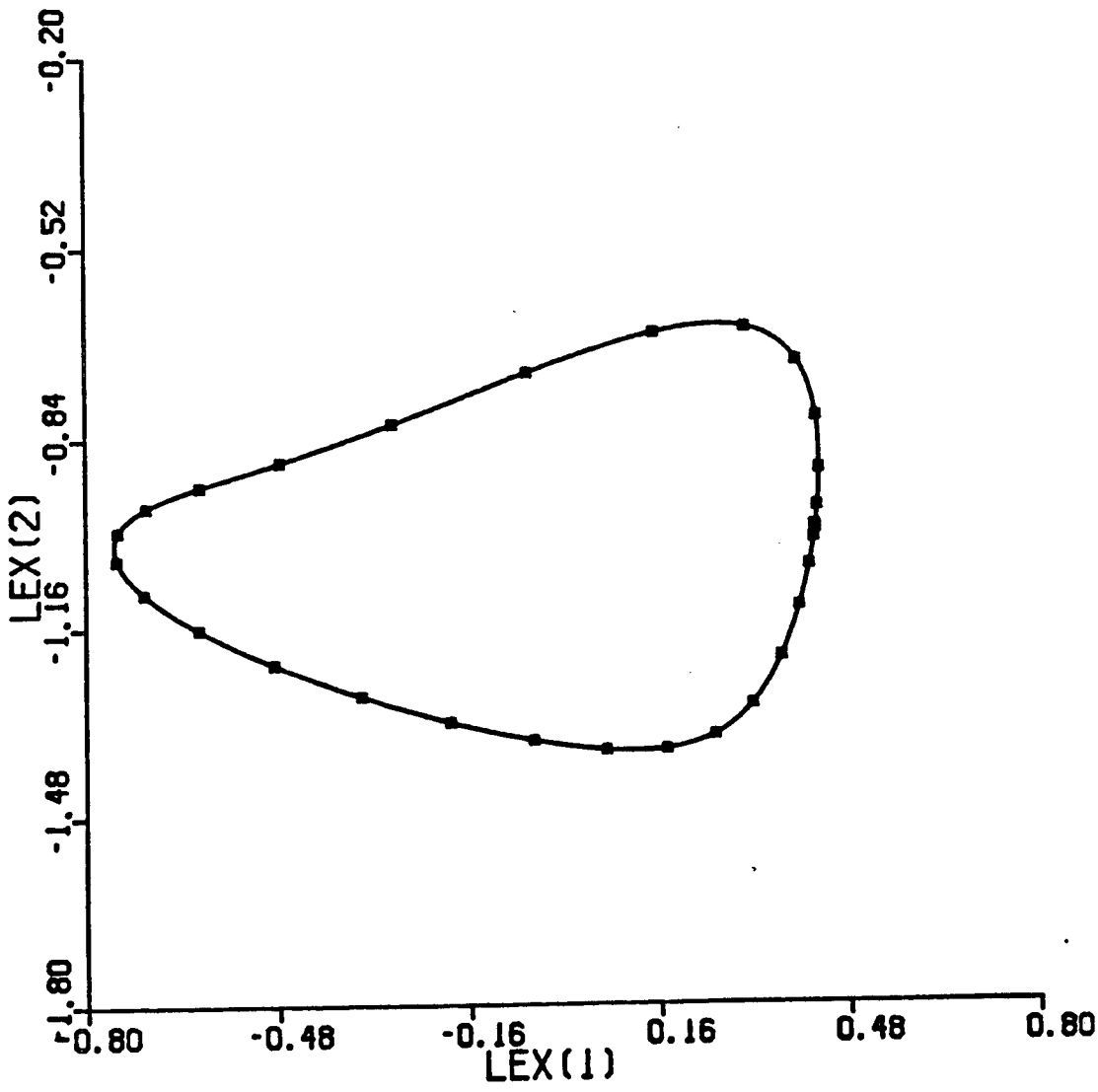
View of circle 1 using the self-learned direct position kinematics after 400 training paths



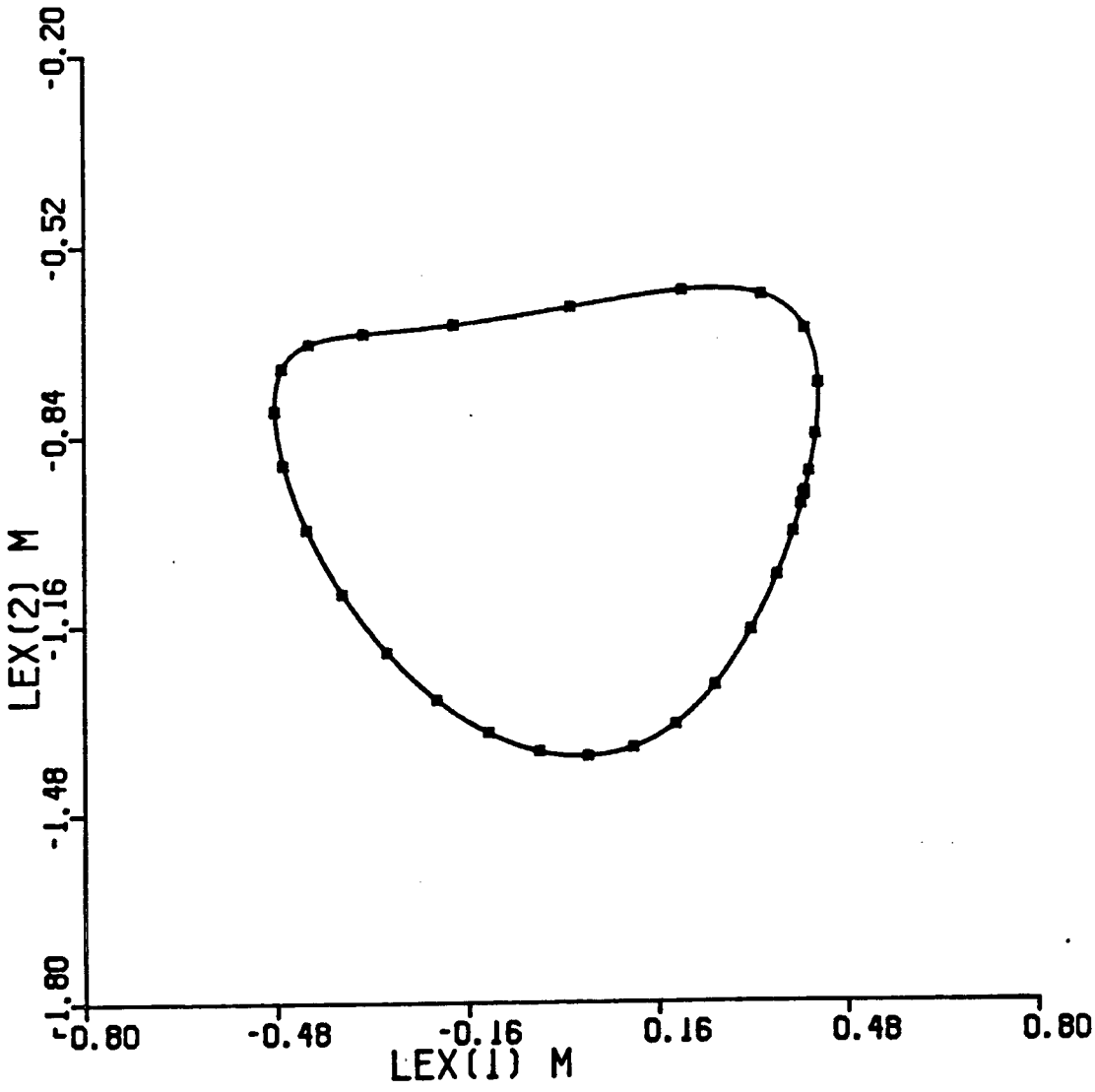
View of circle 1 using the self-learned direct position kinematics after 600 training paths



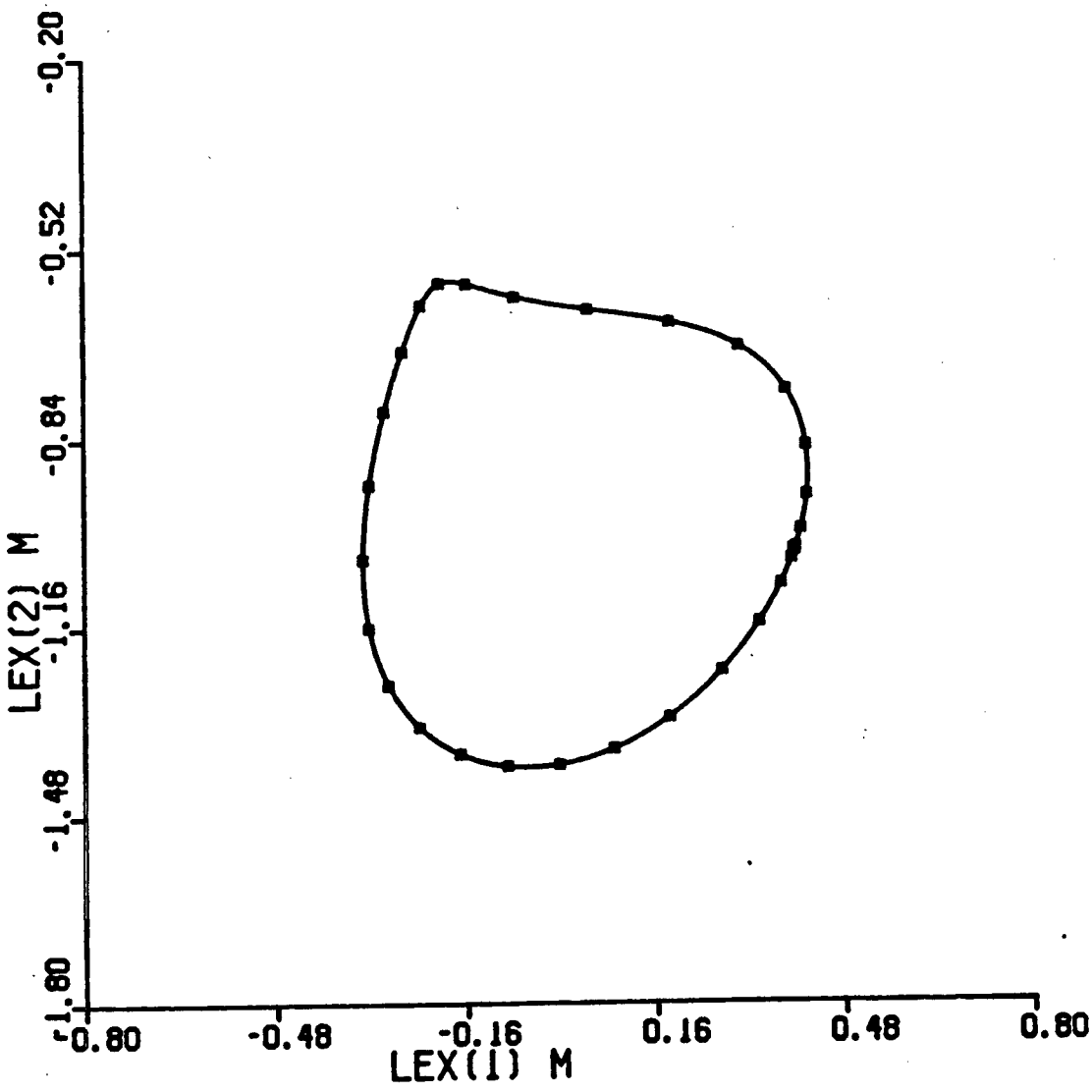
View of circle 1 using the self-learned direct position kinematics after 800 training paths



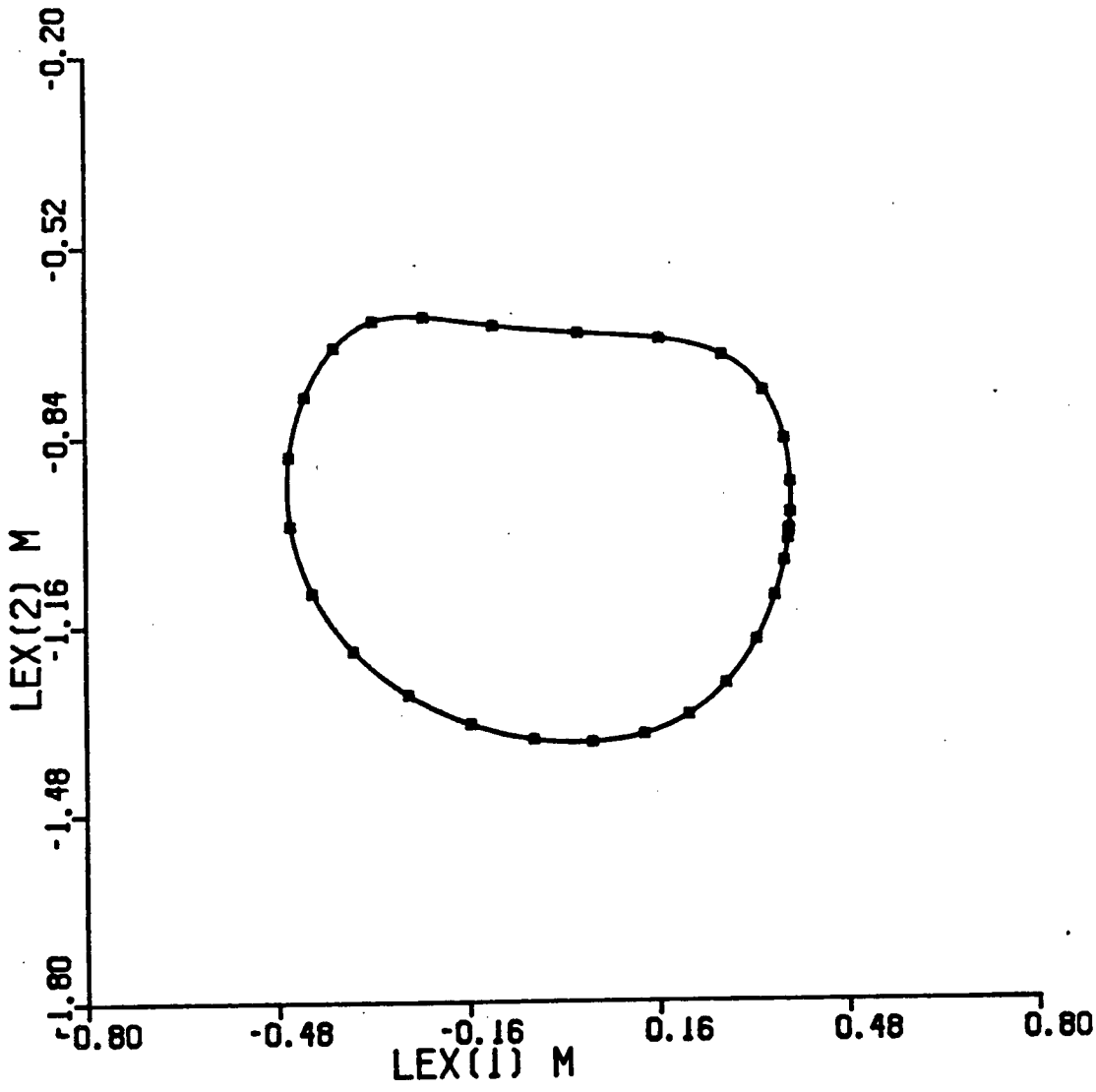
View of circle 1 using the self-learned direct position kinematics after 1000 training paths



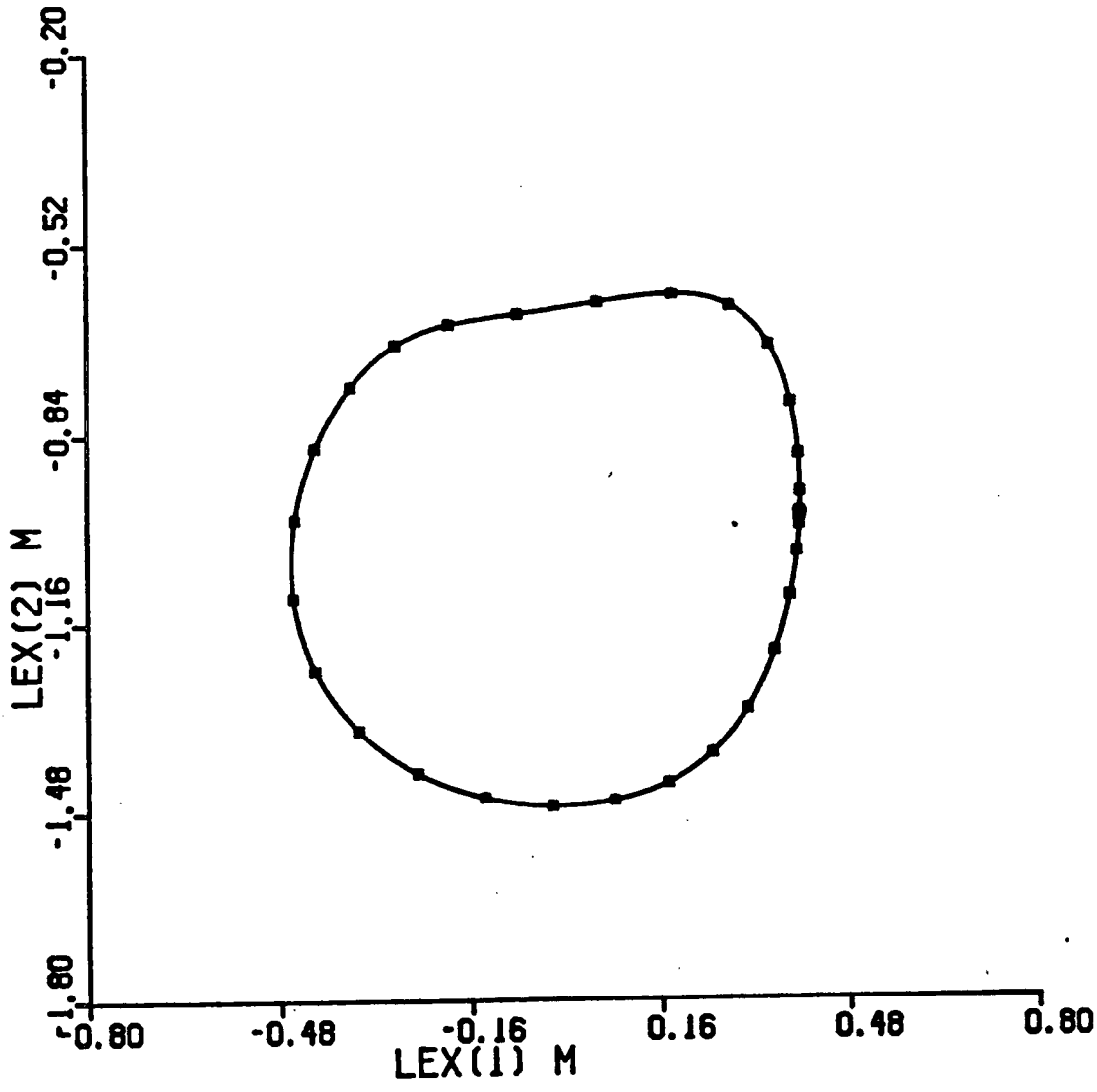
View of circle 1 using the self-learned direct position kinematics after 1400 training paths



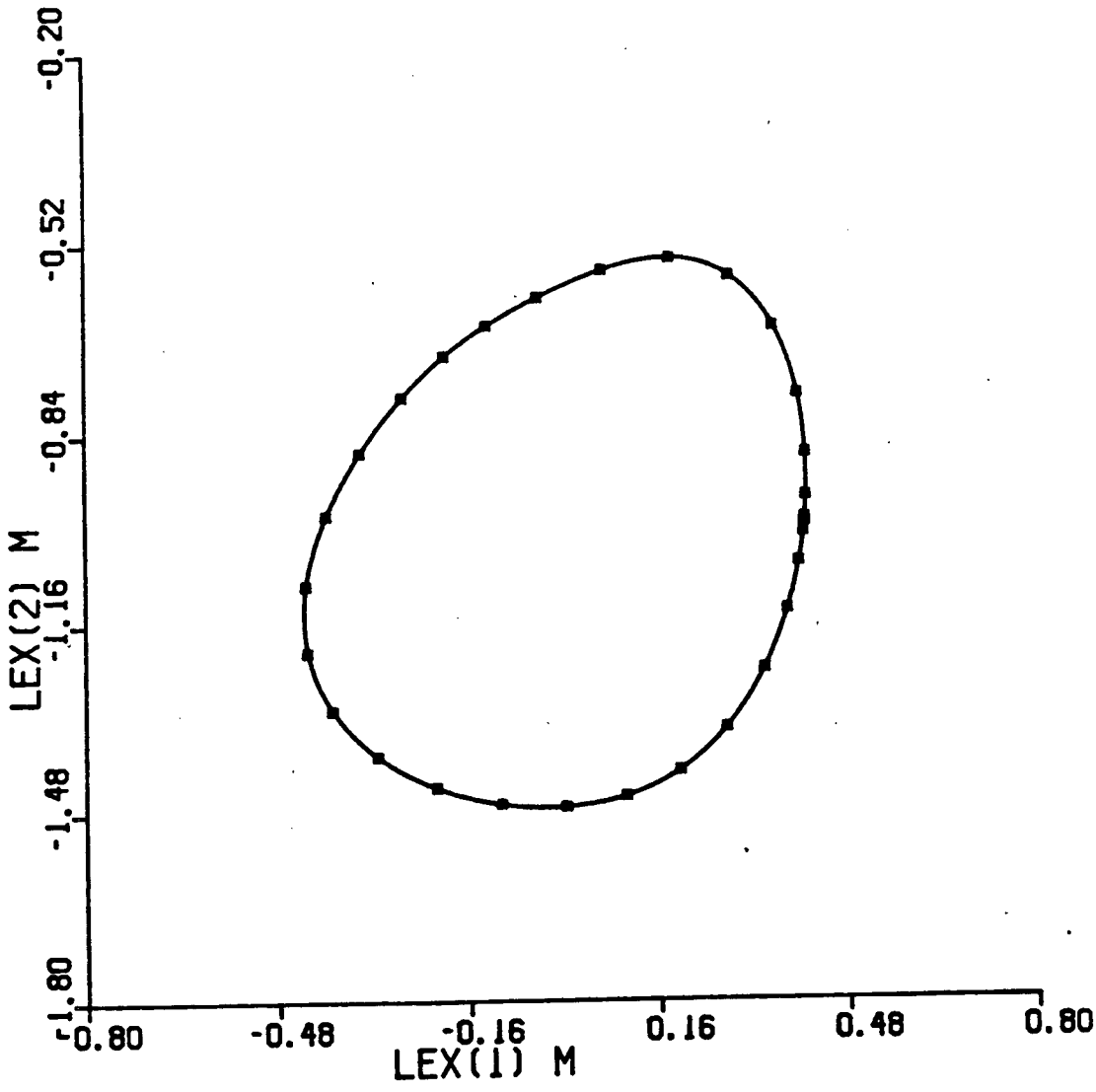
View of circle 1 using the self-learned direct position kinematics after 1800 training paths



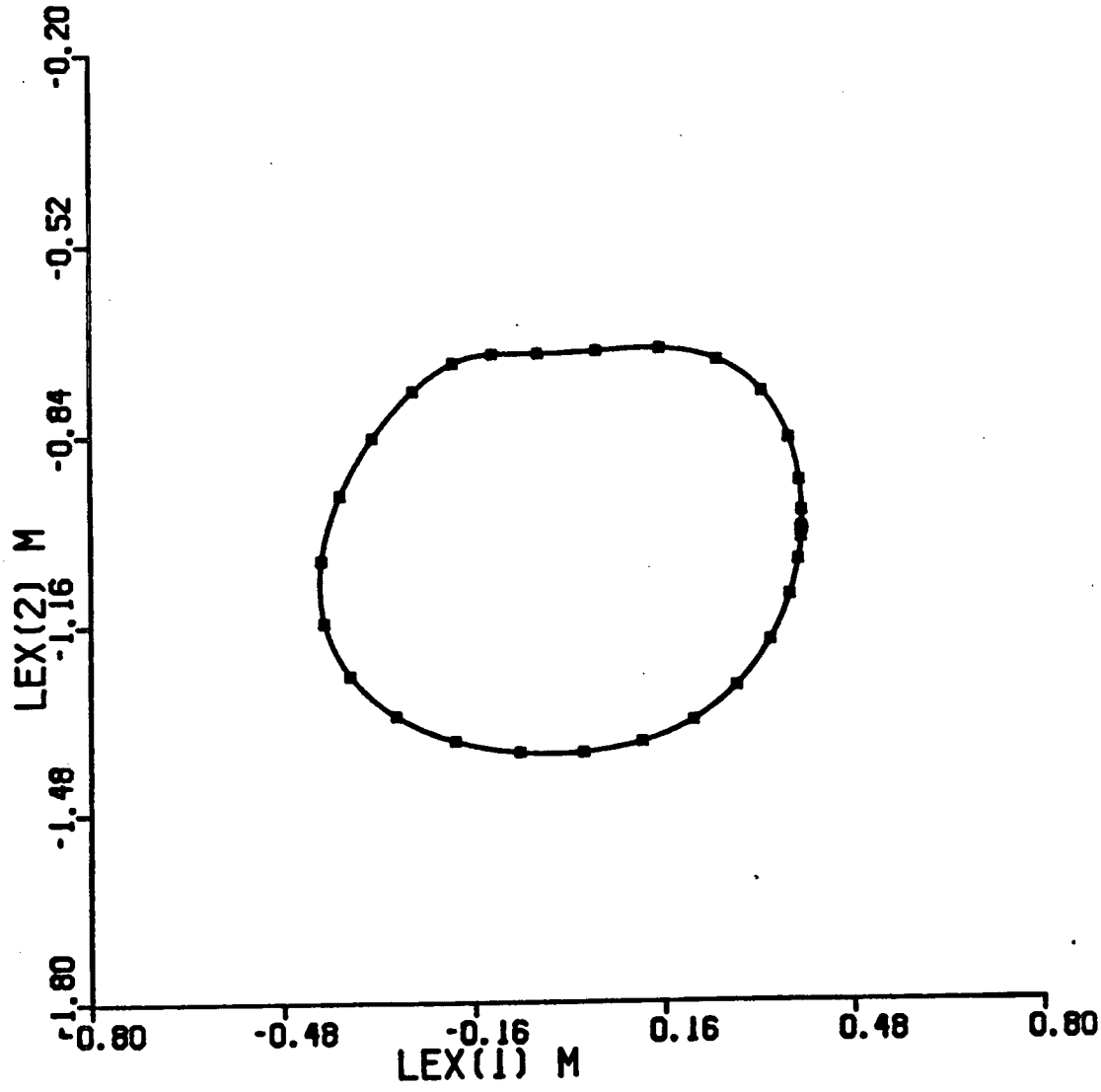
View of circle 1 using the self-learned direct position kinematics after 2200 training paths



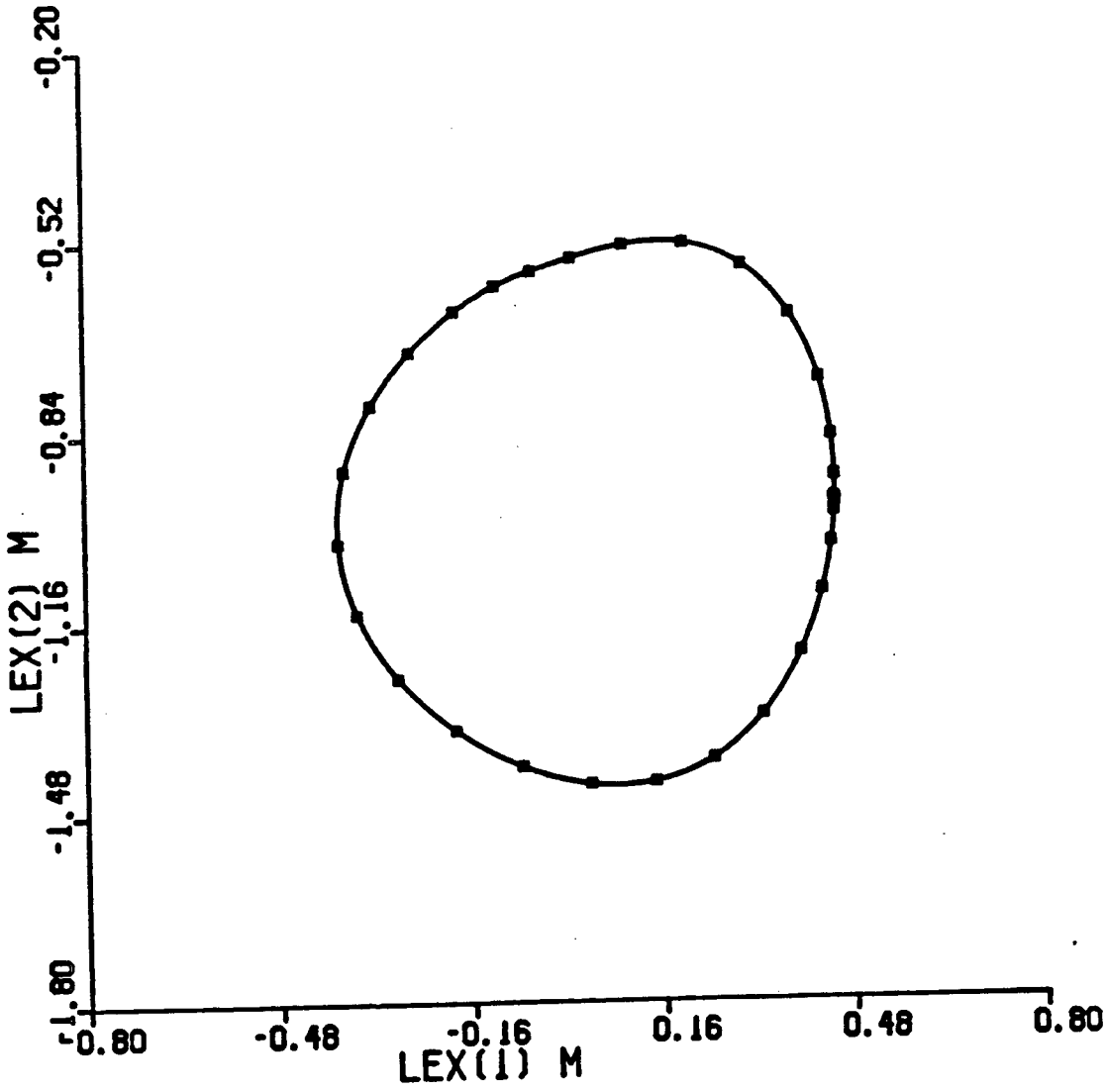
View of circle 1 using the self-learned direct position kinematics after 2600 training paths



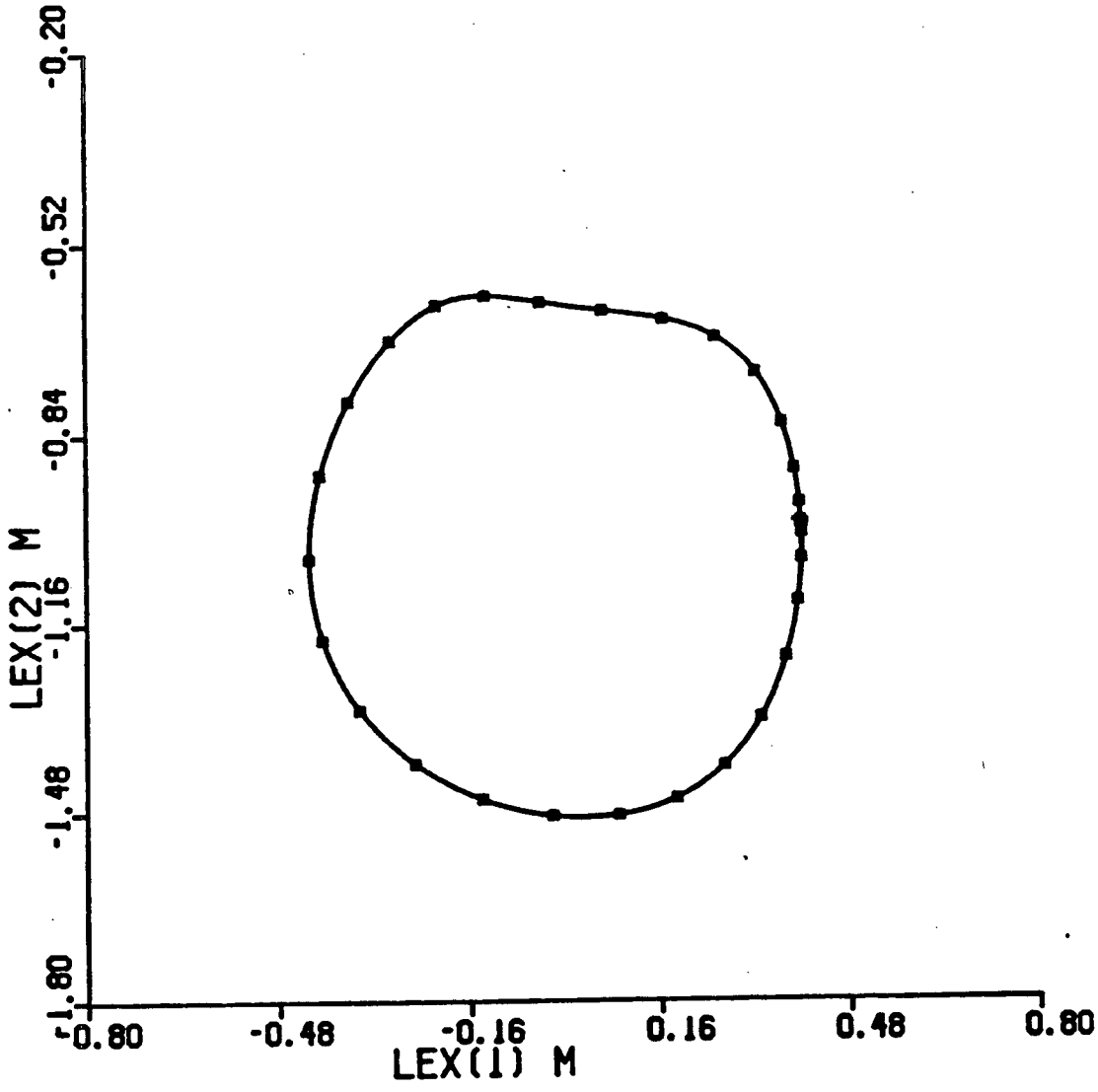
View of circle 1 using the self-learned direct position kinematics after 3000 training paths



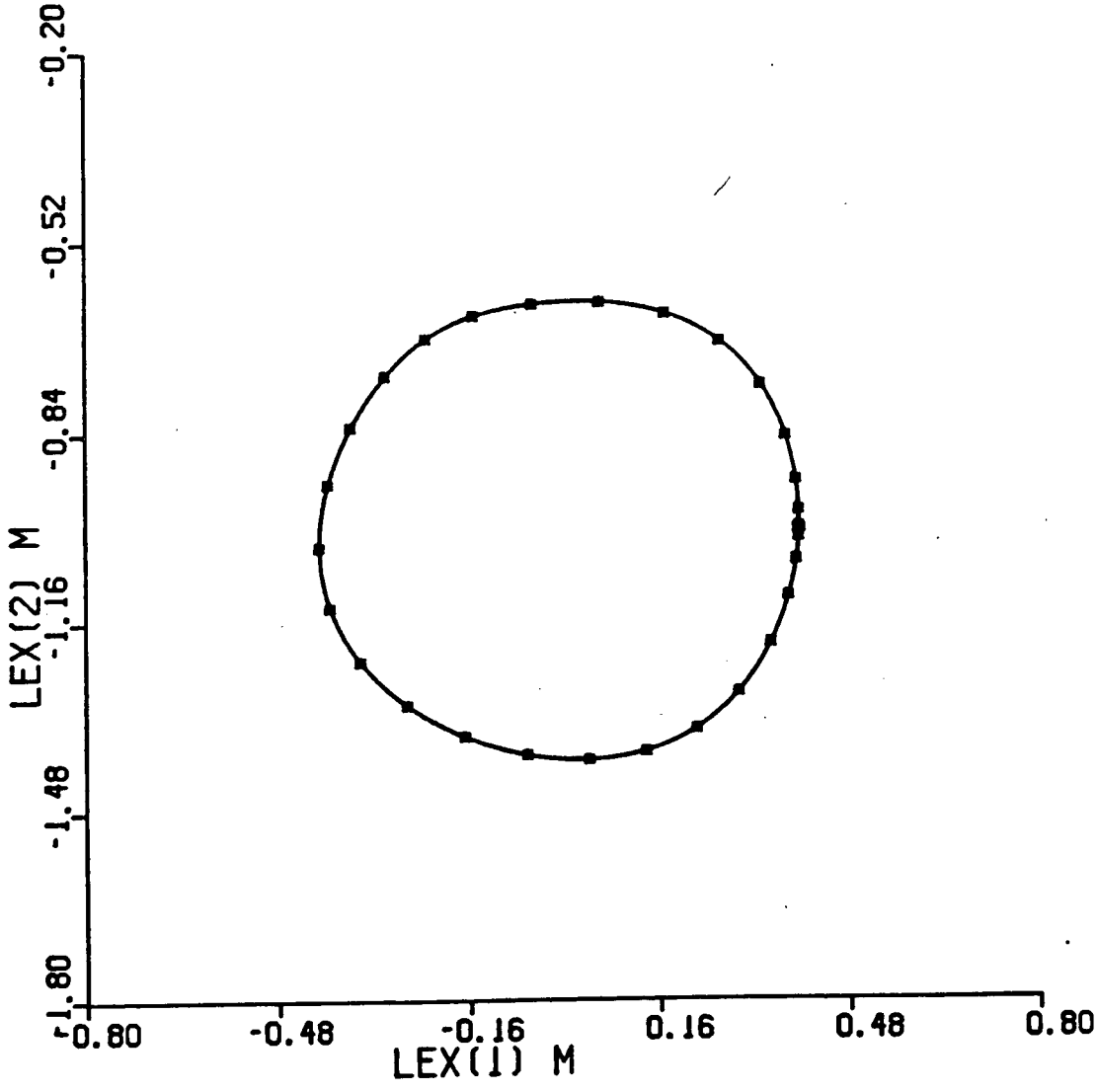
View of circle 1 using the self-learned direct position kinematics after 3400 training paths



View of circle 1 using the self-learned direct position kinematics after 3800 training paths



View of circle 1 using the self-learned direct position kinematics after 4200 training paths



View of circle 1 using the self-learned direct position kinematics after 4600 training paths (final 400 with smoothing)