(2)

WL-TR-92-1122

DRIVE-REINFORCEMENT LEARNING SYSTEM APPLICATIONS

Daniel W. Johnson, Ph.D.
Martin Marietta Aero & Naval Systems
103 Chesapeake Park Plaza
Baltimore, Maryland 21220

July 1992

FINAL REPORT FOR 01/01/89 - 07/01/92

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AFB, OHIO 45433-7301

NOTICE

This technical report has been reviewed and is approved for publication.

James S. Morgan

James S. Morgan
Project Engineer
Advanced Systems Research Section
Avionics Directorate
Wright Laboratory

[illegible]

William R. Baker, Acting Chief
Advanced Systems Research Section
Avionics Directorate
Wright Laboratory

[illegible]

Charles H. Krueger, Jr
Chief
Systems Avionics Division
Avionics Directorate
Wright Laboratory

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 31 July 1992	3. REPORT TYPE AND DATES COVERED FINAL REPORT Jan 1989 - July 1992

4. TITLE AND SUBTITLE Drive - Reinforcement Learning System Applications	5. FUNDING NUMBERS CF33615-89-C-1023 PE-62204 PR-2003 TA-05 WU-56
6. AUTHOR(S) Daniel W. Johnson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Martin Maretta Aero & Naval Systems 103 Chesapeake Park Plaza Baltimore, MD 21220	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Avionics Directorate Wright Laboratory (WL/AAAT) Air Force Material Command Wright-Patterson AFB, OH 45433-6543 James S. Morgan (513-255-7650)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-92-1122
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The application of Drive-Reinforcement (D-R) to the unsupervised learning of manipulator control functions was investigated. In particular, the ability of a D-R neuronal system to learn servo-level and trajectory-level controls for a robotic mechanism was assessed. Results indicate that D-R based systems can be successful at learning these functions in real-time with actual hardware. Moreover, since the control architectures are generic, the evidence suggests that D-R would be effective in control system applications outside the robotics arena.

14. SUBJECT TERMS Drive-Reinforcement Learning; Neural Network Controllers; Robotics; Manipulator Kinematics, Dynamics and Control			15. NUMBER OF PAGES 70
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Contents

Figures

Preface

The work presented here was supported by the United States Air Force, Wright Laboratory, under a three-year contract to experimentally evaluate the use of the Drive-Reinforcement (D-R) algorithm for the unsupervised learning of manipulator control functions.

The purpose of this report is to present some of the highlights of this work. Literally thousands of experiments were performed over a three-year period in order to gain a basic understanding of the Drive-Reinforcement learning mechanism. We do not attempt to describe all these studies in this final report, but we do want to convey a sense of the trail that we blazed. More importantly, we want to describe what we have learned about the nature of the D-R algorithm. This understanding is crucial for fast, reliable implementation of D-R in on-line learning applications, and is required if one expects to make future refinements in its algorithmic structure.

I would like to express gratitude for the very fine work of all the members of the technical staff who supported the D-R efforts at Martin Marietta. Dr. Von A. Jennings was responsible for the initial work and deserves special recognition. He is to be commended for being responsive to the needs of the D-R research community through his persistent efforts to strictly adhere to the theoretical intent of the D-R learning mechanism and through his encouragement of others to do the same. Dr. John J. Murray was responsible for overseeing the development of both our simulation software and the software used in our real-time systems. Without his efforts at rapid prototyping and rapid transitioning of software from simulation to real-time code, we would not have had time to achieve the results presented here. Additional thanks are extended to Dr. Gilbert H. Lovell for his technical assistance on control system architectures, Ms. Karen A. Goodrich for her software design and implementation support, and Mr. Thomas P. Andreychek for building the mechanical and electrical portions of our latest test beds.

1. Introduction

Drive-Reinforcement is a neuronal model developed by Klopf (1988) and his research team at Wright Laboratory as part of their in-house basic research program in machine intelligence. It is similar to other neuronal models in that its output is expressed in terms of a weighted sum of its inputs. Moreover, as in other models, the D-R neuron can be used as a basic processing unit and building block for a general network topology formed by connecting the input of some neuronal elements to the output of others. It is different from other neuronal models in that its learning mechanism has a unique structure. For example, the D-R neuron requires temporal relationships between training samples, and computes weight changes that are proportional to changes in the neuronal output and proportional to filtered values of weighted changes in neuronal inputs. Many neuronal models do not require temporal relationships between training samples, and compute weight changes that are proportional to the error between desired and actual values for the neuronal output. These latter models require the use of an 'external' training signal so that explicit function matching can be performed: weights are adjusted until the actual neuronal output approaches the desired neuronal output. In contrast, the D-R neuron does not require a desired neuronal output signal for its weight adjustment algorithm, but it does require a special neuronal input that acts like an 'internal' training signal. Thus, the interpretation and usage of the input and output signals for the D-R neuron is different from that of many other neuronal models. Moreover, the derivative nature of the D-R learning mechanism puts it in a special class of algorithms (Sutton and Barto, 1990) which perform a matching of function-derivatives versus a matching of explicit function values. Thus, D-R is a complementary neuronal model to those noted above, and this suggests that there may be some applications for which the use of D-R would be preferred.

D-R may be useful at all levels in the hierarchy of control system tasks from actuator control to pattern recognition and mission planning. Our ultimate goal is the development of machine intelligent systems that can learn to perform at each level. Our belief is that a biologically based, evolutionary approach to these problems is required in order to handle the complexity and adaptability requirements for the associated systems. Once we fully understand how D-R can be implemented and "grown" to handle increasingly complex problems, we believe it will become increasingly useful as a basis for machine intelligence. A natural starting point for this understanding is the application of D-R to low-level, nonlinear control problems since these problems form the natural building blocks for the higher-level control functions.

The use of D-R in control system applications offers several advantages over standard nonlinear control techniques, especially when D-R is placed in an unsupervised learning context. For example, it is often inadequate, difficult and costly to use standard model-based control techniques when the mathematical description for the controlled physical system and its environment are either very complex, only partially known or time-varying. Complex systems have high development costs due to the time required to analyse, design, implement and test the mathematical models incorporated in the associated controller. Moreover, controllers for complex systems are often expensive to modify if the configuration of the system is altered. Partially-known systems and time-varying systems can lead to reduced and perhaps unacceptable performance due to the unmodeled dynamic and kinematic relationships and due to the effects of parameters that change unpredictably over time. Thus, the motivation for the use of D-R in an unsupervised control system architecture is clear. Potentially D-R could lower development costs due to a reduction in controller complexity. In addition, by learning to compensate for unmodeled effects and by continuously adapting to time-varying system characteristics, D-R could increase system performance and adaptability.

Our investigation of the application of D-R concentrated on its use in robotic control problems. The nature of the investigation was planned to be experimental and not theoretical, but we found that a marriage of both theory and practice was crucial to understand and successfully apply the algorithm. Our studies were generally begun in simulation prior to their implementation and testing in hardware because this saves time and promotes safety. Despite this practice, we started transitioning to hardware early in the program, and backed off after only meager success and many failures to achieve stable operation of the equipment. Thus, our first transitions to hardware were premature, but they ultimately helped progress by forcing an increase in both simulation activity and theoretical understanding about our algorithmic implementations.

The organization of this report is as follows. In Chapter 2, we describe the D-R neuron, its learning mechanism, and our procedure for processing sensed and commanded data into neuronal input signals. In addition, we illuminate essential features of the D-R learning algorithm by discussing its convergence properties. In Chapter 3, we survey our candidate control structures, from those used in our early experiments to those used in our latest experiments, and we motivate the progression between them based on the theory from the prior chapter. In Chapter 4, we discuss the experimental set-up and present detailed results from our successful studies of the on-line learning of servo-level and trajectory-level controls for robotic systems. Finally, Chapter 5 contains a restatement of principal results and conclusions as well as our recommendations for additional work.

2. Drive-Reinforcement Learning

Conditioning is a process by which an animal experiences and learns temporal and spatial associations among signals or events. These learned associations can include relationships among stimuli or between a stimulus and a response. For example, in the well-known Pavlovian or classical conditioning experiments, a relationship is established between two stimuli: an unconditioned stimulus (US), defined as a stimulus that is innately reinforcing; and a conditioned stimulus (CS), defined as a stimulus whose effect on the response can be altered by conditioning. Typically, the CS in these experiments is presented to the animal in temporal proximity to the US until the CS is capable of eliciting a response sometimes approximating that elicited by the US alone. Hence, the US serves as a training signal which is rendered meaningful by an innate mechanism. The CS is a signal that, through training, becomes a predictor of the US.

Conditioning experiments have yielded a wealth of information about animal learning. Several conditioned stimuli are often used in the same experiment to produce effects that are more interesting and subtle than those possible with only a single CS. Hence, conditioning can be a complex process.

To help understand, explain and predict conditioning phenomena, mathematical models of these learning processes have been formulated and tested against the available experimental data. Drive-Reinforcement, a neuronal model of classical conditioning, is one example. It uses a model of a single neuron to produce effects analogous to the behavior of whole animals in classical conditioning experiments. The modeled inputs to the neuron are: 1) conditioned stimuli mediated by synapses with variable efficacy, and 2) unconditioned stimuli mediated by synapses with fixed efficacy. It is the plasticity of the synaptic connection for each CS and the non-plastic behavior of the synaptic connection for each US which allows for learning and for the storage of acquired knowledge. In this structure, a learning mechanism scales the contribution of each CS to the neuronal response while the contribution of each US to the neuronal response remains fixed. In this way, the response to a CS can change through conditioning while the response to a US acting alone can not.

Although the conditioned and unconditioned stimuli used in classical conditioning experiments may be characterized as binary functions (e.g., either a signal is present or it is not), they may also be viewed as real-valued. As an example, consider defining a CS input to a D-R neuron as the output of a low-level sensory neuron. The sensory neuron may receive inputs relayed from several receptors, each being capable of responding to stimuli encountered in some spatially localized, multidimensional section of an associated sense organ. The region formed by the union of these potentially overlapping sections is known as the sensory neuron's receptive field. Because the same stimulus could simultaneously reach more than one receptor site, the output of the sensory neuron can also vary with the location of the stimulus within the receptive field. Hence, CS-triggered signals in a model would have to be real-valued to capture this behavior.

Each CS and US in our control studies will be treated as a real-valued function of the sensed and commanded signals that are communicated in the physical system under control. Although the US will be computed directly from these quantities, the CS will be computed in a way which is analogous to the simplified sensory neuron processing described above. In particular, each CS will have associated with it a 'receptive field' type subset of the space describing the range of the sensed and commanded data. The CS will be identically zero for any signal combination within its receptive field and otherwise generally nonzero and dependent on the location of the data relative to the interior of its receptive field. The motivation behind this is simply that we would like each CS to respond to a limited region within the range of the data so that they can selectively represent features of the data for presentation to the D-R neuron. These features would generally be more useful than the raw signal values as potential predictors of the US.

This chapter is divided into three sections: Section 2.1 describes the D-R neuron as it was used in our work; Section 2.2 details the receptive field structures that decompose sensed and commanded quantities into the conditioned stimuli presented as neuronal inputs; and Section 2.3 discusses what we have learned about the convergence of the D-R learning mechanism.

The following notations and conventions are used in this chapter. Unless otherwise indicated, all time-dependent functions have the form $v:\mathfrak{I}^+ \rightarrow \mathfrak{R}$ where $\mathfrak{I}^+$ denotes the set of positive integers and $\mathfrak{R}$ denotes the set of real numbers. For such functions we let $|v(t)|$ denote the absolute value of $v(t)$, $\Delta v(t) = v(t) - v(t-1)$ and $v(t)^+ = max\{0,v(t)\}$, for all $t \in \mathfrak{I}^+$.

2.1 Mathematical Specification

Our studies are based on a neuronal model which is a slight extension to a special case of the general neuronal model proposed by Klopf (1988). Our intent is to describe it here in a way which illuminates the similarities and differences to the original formulation. Deviations from the original notation were only necessary in order for us to clarify important points in later sections of this report, and were kept to a minimum.

Output Equation

The neuronal input/output relationship that we assumed is similar to that found in Klopf (1988). We too take the view that neuronal signals are real-valued to reflect some measure of neuronal firing frequency, and we express the neuronal output as a function of a weighted sum of neuronal inputs. Here, the neuronal inputs include both conditioned and unconditioned stimuli, and each weight reflects the efficacy of the synapse through which the corresponding input is transmitted with the weights corresponding to excitatory synapses held strictly positive and the weights corresponding to inhibitory synapses held strictly negative.

We specialize the general neuronal input/output relationship in the following ways. First, we present each conditioned stimulus (CS) to the neuronal model through both an excitatory synapse and an inhibitory synapse, analogous to the example found in Klopf (1988). Thus, for the ith CS, denoted by x_i, both an 'excitatory weight' w_i^+ and an 'inhibitory weight' w_i^- are available to the D-R learning mechanism for modification. Second, we present only one unconditioned stimulus (US) to the neuronal model and specify that this US, denoted by u, have a synaptic efficacy equal to one. This is justified since unconditioned stimuli are assumed hardwired with innately determined fixed gains and because we can consequently scale any US and/or combine multiple US sources prior to their introduction into a neuronal model. Each CS could also be scaled, but the associated weights must remain variable and only alterable by the D-R learning mechanism. Neither of the above specializations appear to limit our application of D-R.

Two minor extensions were made to the general neuronal input/output relationship found in Klopf (1988). These, in our opinion, do not change the theoretical intent of D-R because the Drive-Reinforcement learning mechanism is not affected. Moreover, as we will see in Section 2.3, these changes allow for convergence characteristics similar to those cited in Klopf (1988). One change adds flexibility to the way the conditioned stimuli and the unconditioned stimulus are summed in the D-R neuron and the other change is the introduction of a time-dependency to the (originally assumed constant) neuronal threshold, θ. In particular, in the absence of any limiting, the neuronal output (postsynaptic signal level) is given by

$$y(t) = u(t) + \mu \sum_{i=1}^{n} w_i(t)x_i(t) - \theta(t) \tag{2.1-1}$$

where

$$w_i(t) = w_i^+(t) + w_i^-(t), \; i = 1,\ldots,n \tag{2.1-2}$$

is the net weight (gain) associated with the ith CS (a presynaptic signal level) at time t, n is the number of conditioned stimuli presented to the neuron, and $\mu \in \{-1,+1\}$ is an application dependent switch that we set a priori in order to establish a particular relationship between the unconditioned stimulus and the conditioned stimuli (see Section 2.3). Note that for convenience we are using x_i to denote two separate but identical inputs to the D-R neuron; this notation differs from Klopf (1988) where the x_i, $i \in \{1,\ldots,n\}$ are used to represent general neuronal inputs. Equations 2.1-1 and 2.1-2 thus model 2n+1 neuronal inputs since each CS makes two synaptic connections and only one US is used. The neuronal threshold in equation 2.1-1 was often set to zero in our work, but adding the possibility for time-dependent behavior allows one to adjust the neuronal output during learning. As described in Section 2.3, this helps us achieve other desirable, application dependent objectives.

Klopf (1988) defines neuronal drives as weighted presynaptic signals and defines neuronal reinforcers as weighted changes in presynaptic signals. In particular, from equations 2.1-1 and 2.1-2, the acquired (learned) neuronal drives are the products $w_i^+(\cdot)x_i(\cdot)$, $w_i^-(\cdot)x_i(\cdot)$, $i = 1,\ldots,n$ and the only primary (innate) neuronal drive is $u(\cdot)$. Similarly, the acquired neuronal reinforcers are the products $w_i^+(\cdot)\Delta x_i(\cdot)$, $w_i^-(\cdot)\Delta x_i(\cdot)$, $i = 1,\ldots,n$ and the only primary neuronal reinforcer is $\Delta u(\cdot)$.

Output Limiting

Upper and lower bounds on the neuronal output are essential parts of the D-R neuronal output description. They are biologically meaningful since an upper bound corresponds to a maximum neuronal firing frequency and a lower bound greater than or equal to zero avoids the necessity of interpreting negative neuronal firing frequencies. More importantly, a non-negative lower bound is required to produce the correct behavior in conditioned-inhibition experiments since it prevents a CS with a negative association from extinguishing when presented alone (see Klopf, 1988; Sutton and Barto, 1990).

In our earliest work we tried to maintain a strict lower bound of zero or greater for the neuronal output despite the fact that negative control signals were desired. This was achieved by bifurcating the neuronal path using a second identically structured neuron with a US equal to the additive inverse of the first neuron's US. The potential for generating both positive and negative control signals was allowed by subtracting the strictly positive output of the second neuron from the strictly positive output of the first neuron. As an alternative to this method, we also tried adding a constant to the US, prior to its introduction into the neuronal model, to force the neuronal output positive. By decrementing the neuronal output, after limiting, by the same constant, both positive and negative control signals could be generated without violating the zero lower bound on neuronal output. Neither approach appeared to benefit the learning process, however, for our applications. As a result, efforts along these lines were dropped in favor of simply adopting a "negative threshold" value for neuronal output, thereby allowing the neuronal output to take on negative values. Although this worked well in our work, it is generally not justified, and a more thorough investigation of output limiting is required.

Bounds on the neuronal output have also been important in our most recent experimental work, but only from a practical point of view. Since unbounded outputs are generally unsafe in a real-time environment and since bounded outputs will often provide the system with a chance to recover from short-term divergent behavior, it is generally prudent to bound the neuronal output from below and above. We have found, however, that the algorithm generally performed better when the limits were not achieved. In fact, we usually set the upper and lower limits such that they were encountered only if the system reached a point of instability. Because neuronal limits were not encountered in successful runs, they are not critical to understanding why D-R works in our applications. In fact, we do not show bounds in equation 2.1-1 as this would only obscure the important relationships found in this expression.

Learning Mechanism

The neuronal learning mechanism that we assumed in all our work is identical to that found in Klopf (1988). Given $\varepsilon > 0$, the excitatory and inhibitory weights are adjusted according to

$$w_i^+(t+1) = \mathit{max}\left\{ \varepsilon,\ w_i^+(t) + \Delta y(t)\sum_{j=1}^{\tau} c_j\, |w_i^+(t-j)|\, \Delta x_i(t-j)^+ \right\},\ w_i^+(0) = \varepsilon,\ i = 1,\ldots,n$$

(2.1-3)

$$w_i^-(t+1) = \mathit{min}\left\{ -\varepsilon,\ w_i^-(t) + \Delta y(t)\sum_{j=1}^{\tau} c_j\, |w_i^-(t-j)|\, \Delta x_i(t-j)^+ \right\},\ w_i^-(0) = -\varepsilon,\ i = 1,\ldots,n$$

where τ is the longest interstimulus interval over which delay conditioning is effective, and $c_j > 0$ is a learning-rate constant proportional to the efficacy of conditioning when the interstimulus interval is equal to $j \in \{1,\ldots,\tau\}$. Note that $w_i^+(t) \geq \varepsilon$ and $w_i^-(t) \leq -\varepsilon$, $i=1,\ldots,n$ for all $t \in \mathfrak{Z}^+$, so that the excitatory weights remain positive and the inhibitory weights remain negative. In addition, since $w_i(0) = w_i^+(0) + w_i^-(0) = 0$, $i = 1,\ldots,n$, the acquired drives do not contribute to the initial neuronal response.

The sums in these expressions are finite impulse response filters which have the effect of smoothing and scaling the data given by the products $w_i^+(\cdot)\Delta x_i(\cdot)^+$, $w_i^-(\cdot)\Delta x_i(\cdot)^+$, $i = 1,\ldots,n$.

They are responsible for enabling learning during each of the τ time intervals following any strictly positive change in any presynaptic signal level, and are responsible for disabling learning otherwise. This is because the sums are always nonnegative and every $\Delta x_i(t)>0$, $i\in\{1,...,n\}$ contributes to its associated filter value for τ time intervals. Hence, a synapse is 'eligible' (see Klopf, 1988) for modification during each of the τ time intervals following any strictly positive change in its associated presynaptic signal level, and is ineligible for modification otherwise. If a synapse is eligible, then either the resulting change in its synaptic efficacy is proportional to $\Delta y(t)$, the change in the postsynaptic signal level, or a weight limit is achieved and the synaptic efficacy is pegged at the limit.

Our only modification to equations 2.1-3 was the addition of an upper bound on the weight values. As in the case of the bounds for the neuronal output equation, this change was mostly for convenience and safety. In practice, the upper bound on weight values was only achieved when instabilities occurred. As a result, explicitly writing these bounds in the above equations would obscure the true nature of the D-R learning mechanism, and for this reason we do not express them in equations 2.1-3. On the other hand, there may be some theoretical significance to the inclusion of an upper bound on synaptic efficacy. See Section 2.3 for details.

2.2 Receptive Field Structures

Our receptive field structures are used to transform sensed and commanded data into conditioned stimuli. The placement of this transformation relative to the D-R neuron is illustrated in Figure 2.2-1. In this figure, $x=(x_1,...,x_n)$ is a time-dependent vector representing the conditioned stimuli presented to the D-R neuron, and $s=(s_1,...,s_d)\in S$ is a time-dependent vector representing the sensed and commanded signals where d is the number of signals and S is the space of all possible signal inputs. Examples of sensed quantities include the position, velocity and acceleration of an object, the distance to an obstacle, the frequency of a vibration, the temperature and humidity of a room, and any particular features of our environment that we need to extract or directly measure for purposes of on-line learning and control. Commanded data often includes the desired time histories of the sensed quantities or higher level goals and objectives for system behavior.

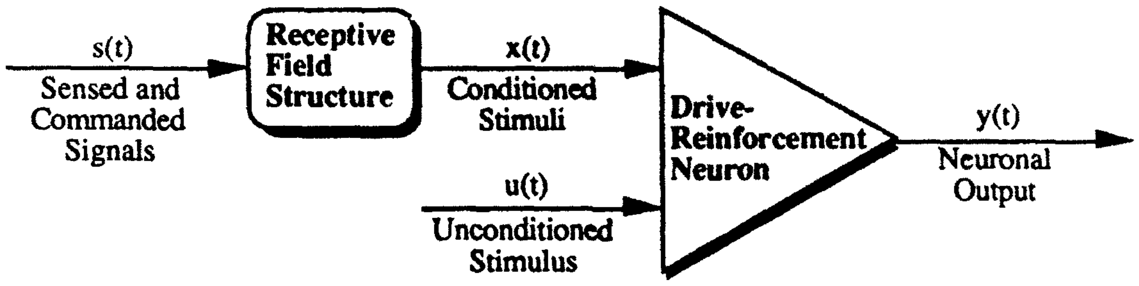
s(t)
Sensed and Commanded Signals
Receptive Field Structure
x(t)
Conditioned Stimuli
u(t)
Unconditioned Stimulus
Drive-Reinforcement Neuron
y(t)
Neuronal Output

Figure 2.2-1. The D-R Neuron and Receptive Field Structure.

The conditioned stimuli will generally be multivariable, nonlinear functions of s. This is because typical control system applications require the representation of complex mathematical relationships, and we will use the weighted sum of these neuronal inputs, as computed by the neuronal model, to describe them. Linear, decoupled functions of s are a special case of our general transformations, but their usage would be equivalent to passing sensed and commanded data directly to the neuron and would have limited value in the work we present here.

When expressed in terms of s, the conditioned stimuli $x_i(t)=\bar{x}_i(s(t))$, $\bar{x}_i:S\rightarrow\Re$, $i=1,\ldots,n$ can be viewed as basis functions for an n-dimensional linear space. Thus, the sum in equation 2.1-1 would have the capability to exactly represent any scalar function belonging to the span of the set given by $\{\bar{x}_i(s),\ i=1,\ldots,n:\ s=(s_1,\ldots,s_d)\in S\}$, and could be used to predict or approximate the primary drive u. Approximation accuracy is improved by increasing n and by selecting nonlinear basis functions which are more representative of a class of functions to which u belongs. In this way, highly nonlinear functions may be represented in our linear space.

The 'support' of each basis function is a subset of the input space S defined by the set $sup(\bar{x}_i)=closure\{s\in S:\ \bar{x}_i(s)\neq 0\}$, $i\in\{1,\ldots,n\}$. It is this subset of S that we call a receptive field. In particular, it is the receptive field associated with the conditioned stimulus x_i, $i\in\{1,\ldots,n\}$. Thus, in analogy to the receptive field of a sensory neuron, the conditioned stimulus $x_i(t)=\bar{x}_i(s(t))$, $i\in\{1,\ldots,n\}$ will be identically zero if $s(t)\notin sup(\bar{x}_i)$, and otherwise generally non-zero and dependent on the location of the s(t) relative to the interior of the set $sup(\bar{x}_i)$. Examples of receptive fields and the corresponding basis functions are provided later in this section.

Generally our receptive fields will be overlapping. They do not have to cover the entire input space because there are regions of S that the physical system will never encounter in practice. Ideally, we would ignore these regions and concentrate the receptive fields in either the most active regions or in regions over which control system parameters vary most rapidly. However, in most applications, we do not have the a priori knowledge required to make these decisions, and must usually be content to uniformly partition the input space so that all subsets of S are equally represented.

We now examine our basis functions in detail, beginning with the consideration of the case in which each CS is a function of only one component of s, and then proceeding to the multivariable case where each CS is a function of multiple components of s. The section is concluded with an examination of the relationship between these transformations and those used in the Cerebellar Model Articulation Controller (CMAC) neural network (Albus, 1975). This is important because much of our earlier work was based on this latter architecture.

One-Dimensional B-Splines

Consider the case where $d=1$ and S is the interval $[a,b]$ for some $a,b \in \Re$. Although there are many linear function spaces from which to choose, we restrict our attention to spaces of relatively smooth piecewise polynomials: the polynomial splines. The order of a polynomial spline, denoted by m, is defined as the number of coefficients required to represent the polynomial pieces. The polynomial pieces are joined at values of $s \in S$ called knots. For example, in our work we select k knots given by $\{a+ih \in S, i=1,\ldots,k\}$, $h=(b-a)/(k+1)$ so that the input space S is uniformly partitioned into k+1 intervals of length h.

Low-order polynomial splines are ideal for real-time control problems because the elements of these spaces are quick to evaluate and easy to manipulate due to their polynomial character. Moreover, every continuous function on a compact interval can be approximated arbitrarily well by polynomial splines of any order provided a sufficient number of polynomial pieces are allowed (Schumaker, 1981). Hence, increasing the number of knots is sufficient to improve the approximation power of the polynomial splines.

We use only the smoothest space of piecewise polynomials of order m with k genuine knots (i.e. the knots would disappear if the polynomial pieces were joined together any smoother). For example, if m=1 the splines are discontinuous (at the knots) and if m>1 the splines are continuous with m-2 continuous derivatives. This is a real linear space of dimension n=m+k. The most attractive basis for this space are the normalized B-Splines (Schumaker, 1981) because they have compact support and are only non-zero over m spline-knot intervals. This implies that the length of the corresponding receptive field is only mh units long, and that the effects of the synaptic weights in our problem can be strongly decoupled. Ultimately, this has a very positive impact on the efficiency of the D-R learning mechanism. Moreover, our assumption of uniform knot spacing leads to substantial savings in the computation of the basis functions because all the basis elements are constructed as translated versions of a single piecewise polynomial function, $N^m:\Re\rightarrow\Re$. In particular, given $s(t)\in S=[a,b]$, the values of the basis elements are computed by

$$\bar{x}_i(s) = N^m\left(\frac{s-a}{h}+m-i\right),\ i=1,\dots,n \tag{2.2-1}$$

where, for example, the N^m for m=1,2,3,4 are

$$N^1(z)=\begin{cases}1, & 0\le z<1\\ 0, & \text{otherwise,}\end{cases}\qquad N^2(z)=\begin{cases}z, & 0\le z\le 1\\ 2-z, & 1\le z\le 2\\ 0, & \text{otherwise,}\end{cases}$$

(2.2-2)

$$N^3(z)=\begin{cases}z^2/2, & 0\le z\le 1\\ -z^2+3z-1.5, & 1\le z\le 2\\ N^3(3-z), & 2\le z\le 3\\ 0, & \text{otherwise,}\end{cases}\qquad N^4(z)=\begin{cases}z^3/6, & 0\le z\le 1\\ -z^3/2+2z^2-2z+2/3, & 1\le z\le 2\\ N^4(4-z), & 2\le z\le 4\\ 0, & \text{otherwise.}\end{cases}$$

Figure 2.2-2 illustrates the basis functions for m=1,2,3,4 which are active over an arbitrary spline-knot interval indicated by the shaded region. Notice that these functions are non-negative valued and that at most only m of them are active at any one time. Moreover, they form a partition of unity: the (non-weighted) sum of the basis functions, evaluated at any input value, is equal to one.

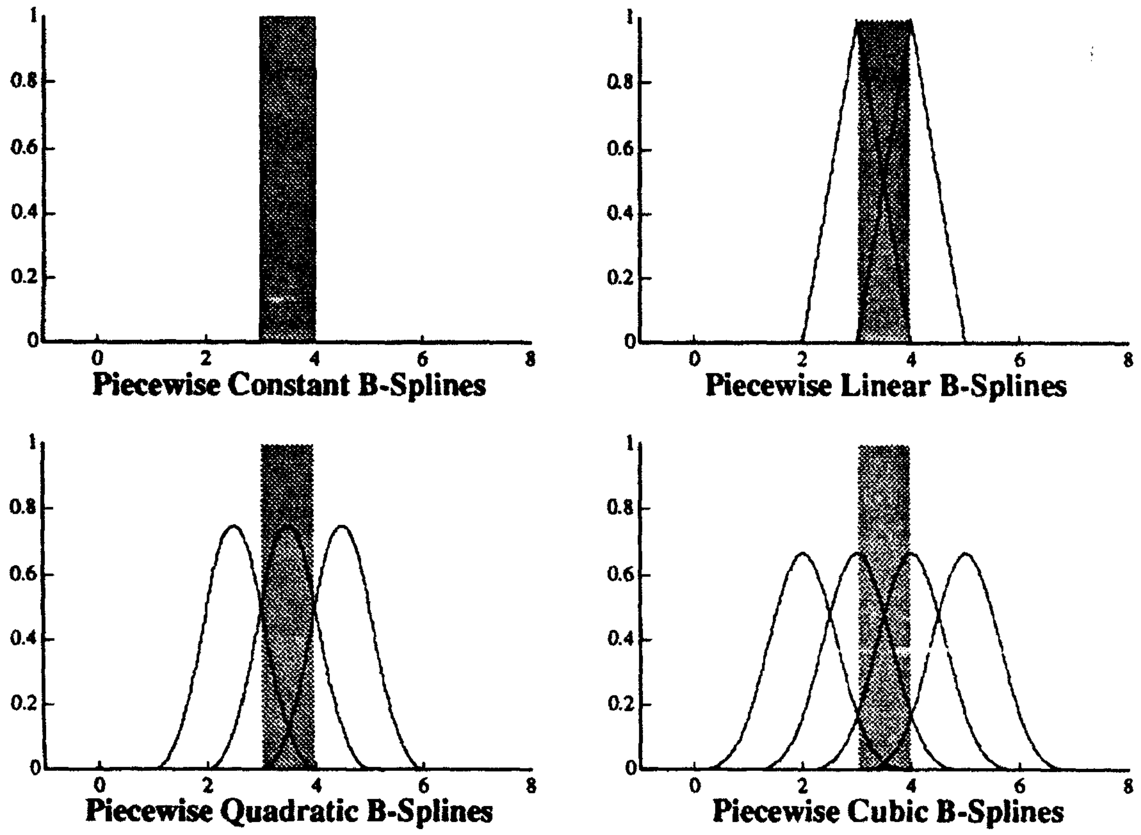

Figure 2.2-2. One-Dimensional Receptive Field Functions.

The general properties implied in Figure 2.2-2 apply to all B-splines, and are inherited by the conditioned stimuli in our problem. In particular, for any $t \in \Im^+$, $x_i(t)=\bar{x}_i(s(t)) \geq 0$, $i=1,\ldots,n$ and $\sum_{i=1}^{n} x_i(t) = \sum_{i=1}^{n} \bar{x}_i(s(t)) = 1$. The small support property allows for the added benefit that at most only m conditioned stimuli are non-zero and active at one time. This decoupling effect helps to isolate features of the sensed and commanded signals so that learning is more efficient.

The above discussion also applies to the case where $d>1$ and the sensed and commanded inputs to the control system are treated as uncoupled quantities. In this event, there would be a different set of conditioned stimuli associated with every component of $s=(s_1,\ldots,s_d) \in S$, and each of these sets would follow the one-dimensional basis function construction outlined in this section. The total number of conditioned stimuli presented to the D-R neuron would then equal the sum of the number in each set.

Tensor-Product B-Splines

For the general multivariable case where the components of s are not independent, we need to construct a space of multidimensional splines. The tensor-product of one-dimensional spaces of polynomial splines (S̆umaker, 1981) is often used for this construction because it preserves many of the algebraic properties of the one-dimensional polynomial splines.

Consider the case where S is the hypercube $\{s=(s_1,\ldots,s_d)\colon\ s_j\in[a_j,b_j],\ j=1,\ldots,d\}$ for some $a_j,b_j\in\Re$, $j=1,\ldots,d$. Since our tensor-product spaces are derived from our one-dimensional spaces, we need to specify parameters which are analogous to those in the previous section. In particular, for $j=1,\ldots,d$: define k_j knots in the interval $[a_j,b_j]$ such that $h_j=(b_j-a_j)/(k_j+1)$ is the spacing between the knots, and let m_j be the order of splines that are representable along the j^{th} coordinate direction. The space of tensor-product polynomial splines is easily constructed as the span of one-dimensional B-spline products. It is a real linear space of dimension $n=\prod_{j=1}^{d}(m_j+k_j)$. Products of one-dimensional B-splines are computationally efficient basis functions for this space because they too exhibit small, compact support. Moreover, due to our uniform grid structure, they are calculated by translations of a single function. Given $s(t)\in S$, the values of these basis elements are computed by

$$\bar{x}_{i_1 i_2 \cdots i_d}(s)=\prod_{j=1}^{d} N^{m_j}\left(\frac{s_j-a_j}{h_j}+m_j-i_j\right),\ i_j=1,\ldots,m_j+k_j,\quad j=1,\ldots,d. \tag{2.2-3}$$

Figure 2.2-3 illustrates the shape of these basis functions for $d=2$ and $m_1=m_2=1,2,3,4$.

The n conditioned stimuli associated with these basis functions at time t are given by $x_{i(i_1,\ldots,i_d)}(t)=\bar{x}_{i_1 i_2\cdots i_d}(s(t))$, $i_j=1,\ldots,m_j+k_j$, $j=1,\ldots,d$ where the mapping $i(i_1,\ldots,i_d)\in\{1,\ldots,n\}$ computes a one-to-one correspondence between the indices of x and $\bar{x}$. They too inherit some of the properties that are passed down from the one-dimensional polynomial splines. In particular, for any $t\in\mathfrak{I}^+$, they are non-negative and form a partition of unity: $x_i(t)\geq 0$, $i=1,\ldots,n$ and $\sum_{i=1}^{n} x_i(t)=1$. Furthermore, due to the small support property, at most only $\prod_{j=1}^{d} m_j$ conditioned stimuli are active (non-zero) at one time.

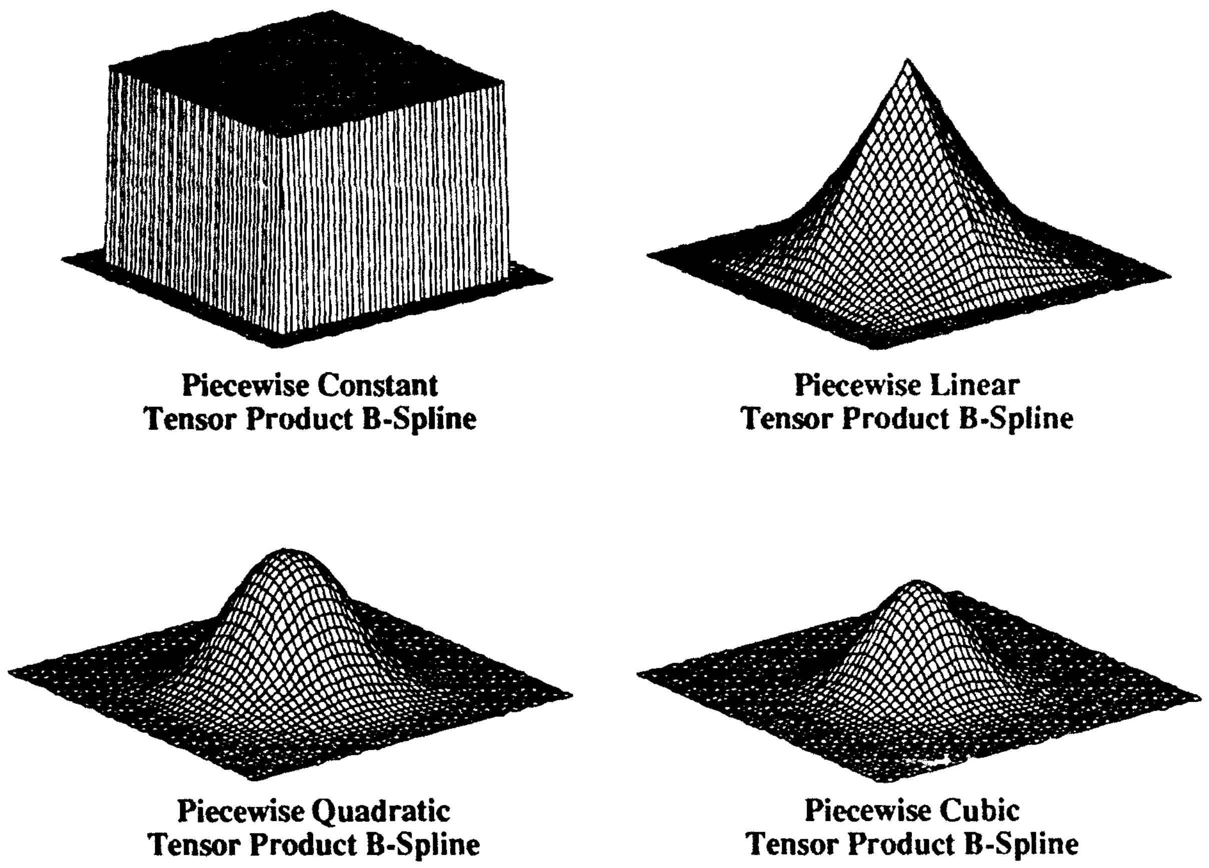

Figure 2.2-3. Two-Dimensional Receptive Field Functions.

CMAC-Based Representations

A deterministic view of the pre-neuronal processing that is performed in the CMAC neural network indicates that it employs the use of receptive field functions similar to our B-splines with $m_j=1, j=1,\ldots,d$. The main difference is that it overlaps layers of these complete basis spaces along the main diagonal of the hypercube S. This has the advantage of adding generalization to this space, but it is the only space presented above without the required overlapping receptive fields.

We stopped using the CMAC-based receptive field functions for the same reason we don't use the piecewise constant spaces presented above. Control problems generally require the representation of smoother functions, and it was more natural to resort to a smoother function space than to use CMAC on a finer grid. Generalizations of the CMAC receptive field functions, based on smooth polynomial splines, have been examined, for example, in Lane, Handelman, and Gelfand (1992), but we did not have the opportunity to fully explore their use with D-R.

2.3 Convergence Properties

When considering applications of D-R, several questions come to mind. For example, what classes of problems can it solve? How is it best implemented in terms of network topology and use of other neuronal structures? Under what conditions and to what 'solution' will the D-R learning mechanism converge? How fast does it learn, can it be made to learn (converge) faster, and can unstable learning be prevented? How does it compare to other methods for solving the same problems?

One accomplishment in our work was the gaining of an understanding of the convergence properties of the D-R learning mechanism. This was essential to the successful implementation of D-R in our application, and it led to an understanding of whole classes of problems which D-R could solve. There was no attempt to develop a rigorous theoretical framework for convergence results, but we now have good experimental insights into what makes D-R work, we have accurate heuristic explanations as to why it works, and we know what it is doing when it does. Moreover, we have found ways to dramatically improve its convergence rate through only slight modifications to the D-R learning algorithm. Though these modifications were not implemented and tested in hardware, we are now convinced that D-R can be used effectively and reliably to solve large classes of engineering problems.

The starting point in the development of D-R was not to solve a particular class of problems, but was to model natural biological learning processes. Thus, in the derivation of the D-R learning algorithm, there were no performance indices being minimized, no functions being approximated, no variables being controlled, and no parameters being estimated. In fact, contrary to the formulation of many other neural network models applied in engineering practice, standard engineering methodologies were not used to develop the D-R learning mechanism. Furthermore, the association between classical conditioning and classical control was not clear at the outset. What 'problem' was the D-R learning mechanism trying to solve? This question had to be answered in the context of typical control engineering problems before successful implementation strategies could be developed.

Heuristic Interpretation

We use the term 'convergence' loosely, and say that the D-R learning algorithm has converged once its synaptic weights have stopped changing. This definition allows us to work backwards, to discover the problem being solved by D-R through an examination of the properties of the 'solution' given by the final weights of a converged algorithm. To derive conditions on these final weights, we begin with equation 2.1-1 and see that the change in the output of the D-R neuron at time t is given by

$$\Delta y(t) = \Delta u(t) + \mu \sum_{i=1}^{n} [w_i(t)\Delta x_i(t) + x_i(t-1)\Delta w_i(t)] - \Delta\theta(t). \tag{2.3-1}$$

This is used in equations 2.1-3 to calculate the weight values at the next sample time. In particular, in the absence of any weight limiting, equations 2.1-2 and 2.1-3 indicate that the new net weight (gain) associated with the i^{th} CS at time t+1 is given by

$$w_i(t+1) = w_i(t) + \Delta y(t)\, \alpha_i(t),\ i = 1,\ldots,n \tag{2.3-2}$$

where

$$\alpha_i(t) = \sum_{j=1}^{\tau} c_j\, [w_i^+(t-j) - w_i^-(t-j)]\, \Delta x_i(t-j)^+,\ i = 1,\ldots,n. \tag{2.3-3}$$

Clearly, $\alpha_i(t) \geq 0$ since $c_j > 0$, $j=1,\ldots,\tau$ and since $w_i^+(t) \geq \varepsilon$, $w_i^-(t) \leq -\varepsilon$ ($\varepsilon > 0$), $i=1,\ldots,n$ for all $t \in \mathfrak{I}^+$. In particular, $\alpha_i(t) > 0$ if and only if $\Delta x_i(t-j) > 0$ for some $j \in \{1,\ldots,\tau\}$, and $\alpha_i(t) = 0$ otherwise. Hence, equation 2.3-2 is also qualitatively accurate for the cases in which we achieve the lower bounds on the magnitudes of the synaptic weights. That is, if $\alpha_i(t) > 0$, then w_i will either increase, decrease or stay the same based on whether $\Delta y(t)$ is positive, negative or zero. Similarly, if $\alpha_i(t) = 0$, then w_i will not change regardless of the value of $\Delta y(t)$. Except for the transitory periods in which weight limits are approached and obtained, equation 2.3-2 is quantitatively accurate as well, and in fact is exact in the limit as $\varepsilon > 0$ approaches zero. Consequently, we may rely on equation 2.3-2 for much of our analysis.

It is clear from the above discussion that D-R involves a temporal notion of convergence. Weight changes can occur at any time t in the set $T=\{t \in \mathfrak{I}^+ : \alpha_i(t)>0$ for at least one $i \in \{1,\ldots,n\}\}$, but do not occur at any t in the complement set $T^c=\{t \in \mathfrak{I}^+ : \alpha_i(t)=0, i=1,\ldots,n\}$. Thus, the D-R learning algorithm converges if and only if there exists a $t_c \in \mathfrak{I}^+$ such that $\Delta y(t)=0$ for all $t \geq t_c$, $t \in T$. This strict notion of convergence is generally not achieved in practice due to noise, inaccurate sensor measurements, algorithmic round-off errors and the failure of receptive field structures to exactly represent the primary drive data. However, it does indicate what must happen qualitatively in any proper implementation of D-R. The algorithm does not control what happens to $\Delta y(t)$ for $t \in T^c$, but seeks to force $\Delta y(t)$ to zero for $t \in T$. As a result, the magnitude of $\Delta y(t)$, $t \in T$ serves as an approximation to a temporal descent function which the D-R learning algorithm must minimize in order to achieve convergence (of the weights).

There are other possible interpretations of this result. For example, one could say that the D-R learning algorithm is correlating changes in the acquired drives with changes in the primary drives by uncorrelating changes in the neuronal output (on the set T) with earlier, positive changes in the neuronal inputs associated with the acquired drives. Alternatively, one could say that the temporal slopes of the acquired drives are being matched to the temporal slopes of the primary drives on the set T. By viewing the primary drive as a function of sensed and commanded signals, $u(t)=\bar{u}(s(t))$, $\bar{u}:S \rightarrow \mathfrak{R}$, one could also present a spatial version of this slope-matching using the function spaces described in Section 2.2, but such an analysis is beyond the scope of this report due to the special directional derivatives that are required to handle the slope discontinuities present in some of the basis functions. In all cases, the algorithm requires that the primary (innate) drive (any neuronal input mediated by a nonplastic synapse) act like an 'internal' training signal. Moreover, in all cases, slightly different functional relationships are formed dependent on the value of the application dependent switch $\mu \in \{-1,+1\}$. In particular, if $\mu=-1$, the relationship is direct, and the weighted sum of the changes in the conditioned stimuli approaches, through D-R learning, the value of the change in the US (primary drive). If $\mu=+1$, the relationship is indirect, and the weighted sum of the changes in the conditioned stimuli approaches the additive inverse of the value of the change in the US. Either way, it is the primary drive which D-R is modeling.

Final Values

Since our strict definition of convergence is generally not achieved in practice, we normally accept approximate solutions. In fact, $\Delta y(\cdot)$ can only be expected to approach zero if there exists weights $w_i^* \in \Re$, $i=1,\ldots,n$ satisfying

$$0 = \Delta u(t) + \mu \sum_{i=1}^{n} w_i^* \Delta x_i(t) - \Delta\theta(t) \tag{2.3-4}$$

for all $t \geq t_c$, $t \in T$. This depends on whether or not the acquired drives can represent signals that have the functional characteristics of the primary drive. Noise and other inaccuracies contribute to this common neural network problem so that weights keep varying after reasonable convergence has been obtained. To correct for these errors, an exponential decay is often applied to learning coefficients. However, this is inappropriate for on-line learning schemes which need to adapt to time-varying properties. Fortunately, in our work, the variations in $\Delta y(\cdot)$ about zero decreased to acceptable levels of error, and no modifications to the D-R learning algorithm were required.

In order to study the final values for the neuronal output, we will assume that the D-R learning algorithm strictly converges. In this case, it is clear that $y(\cdot)$ must be piecewise constant, in the discrete sense, and only allowed to change in value at $t \in T^c$. This implies that given a subset T^* of T containing contiguous values of time greater than t_c, a relationship of the form

$$\rho = u(t) + \mu \sum_{i=1}^{n} w_i^* x_i(t) - \theta(t) \tag{2.3-5}$$

must hold for all $t \in T^*$. Here, $w_i^* \in \Re$, $i=1,\ldots,n$ are the final weights of the converged algorithm and $\rho \in \Re$ is the value of the output of the learning neuron on T^*. Generally, ρ is unconstrained in the original formulation of the D-R learning algorithm whenever the acquired drives can represent arbitrary constant functions on all of T^*. Hence, its final value can be dependent on many factors including the temporal sequencing of neuronal input signals. On the other hand, the value for ρ is unique if constant functions do not have a representation in terms of the acquired drives on T^*.

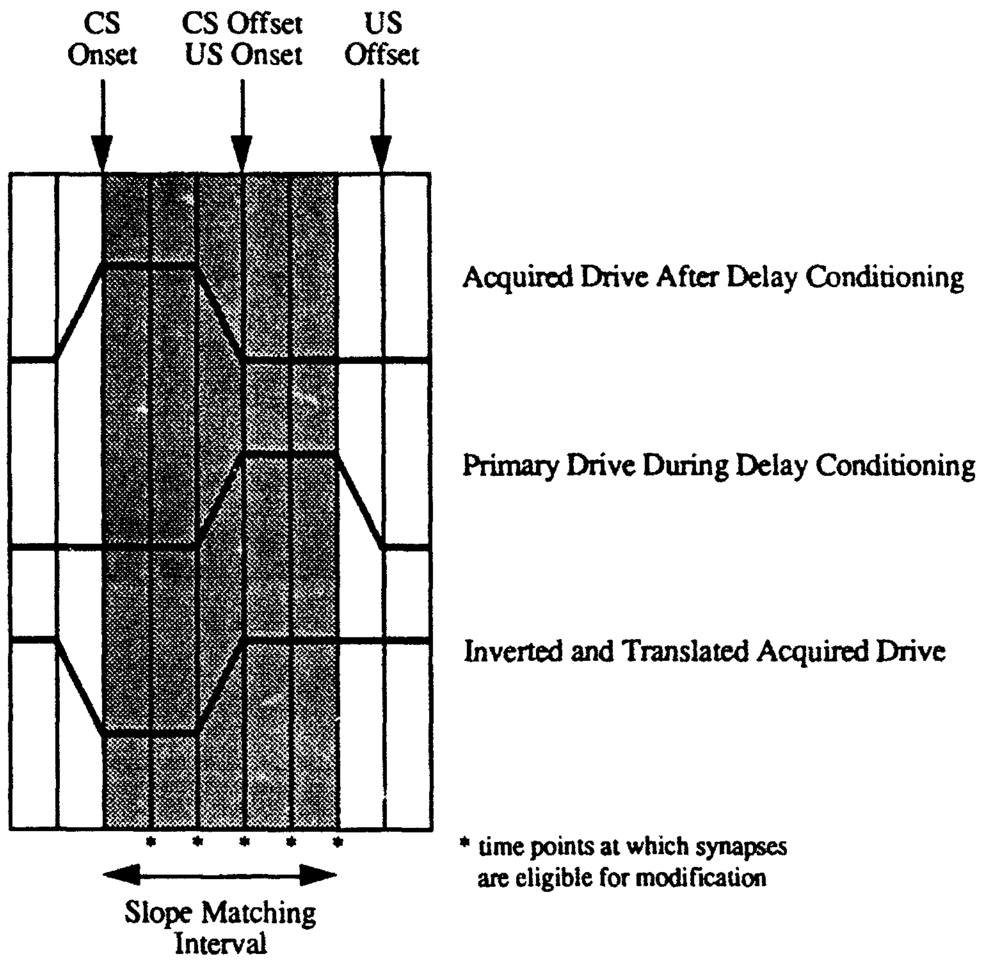
CS Onset
CS Offset US Onset
US Offset
Acquired Drive After Delay Conditioning
Primary Drive During Delay Conditioning
Inverted and Translated Acquired Drive
* time points at which synapses are eligible for modification
Slope Matching Interval

Figure 2.3-1. Convergence in Classical Conditioning Experiments.

An example showing the relevance of the above ideas to classical conditioning is given in Figure 2.3-1 where the neuronal drives are drawn in continuous time as they might be viewed by the D-R learning mechanism and *'s indicate time values in T*. Here, $n=1$, $\mu=+1$ and $\theta(\cdot)=0$ so that the association between the slopes of the acquired and primary drives is learned as an additive inverse relationship, but only in the shaded region. Thus, the CS 'predicts' the US. Alternatively, the relationship may be viewed as direct if we invert the acquired drive and translate it by ρ units, as in the lower graph. Note that ρ is uniquely specified because the acquired drive is not able to represent arbitrary constant functions on the set T*. In particular, since $u(\cdot)$ is a fixed pulse acting alone, ρ must equal the amplitude of $u(\cdot)$. Hence, the final net weight w_1^* is unique and given by equation 2.3-5 at any value of $t \in T^*$ where the CS is acting alone. Classical conditioning relationships are generally more complicated than this example, but similar observations apply.

The above ideas were also important in our most recent control system studies where we set $\mu=-1$ to establish a direct relationship between the slopes of the primary and acquired drives In this work, we selected B-spline basis functions of order $m>1$ and specified that the associated receptive fields cover all of the input space S. Since our basis functions form a partition of unity, a decrease in the value of one basis element leads to an increase in the value of another. Consequently any motion in the system always results in a positive change to at least one CS. Hence, if the system moved, it learned, and since we always kept our systems in motion, T^c was empty and all elements of T^* were contiguous. The downside of having a partition of unity is that constant functions can be represented by the acquired drives on T^*. Thus, ρ becomes a free parameter wnose value is subject to the conditions of the training process. In our servo-level control experiments, we counteracted this by strategically removing basis elements near the origin [illegible] constrain ρ to zero. In our trajectory-level control experiments, we varied the neuronal threshold to constrain ρ to zero. In particular, we assumed that we knew $\overline{u}(s_0)$ and $\overline{x}_i(s_0)$, $i=1,...,n$ for some $s_0 \in S$, and calculated

$$\theta(t) = \overline{u}(s_0) + \mu \sum_{i=1}^{n} w_i(t)\ \overline{x}_i(s_0) \tag{2.3-6}$$

at every time-step. This works well provided values of s near s_0 are encountered during training.

Assuming the D-R learning algorithm strictly converges, it generates only one solution, but even slight modifications to training data can result in a different set of weights. This is because the solution to the problem which D-R is solving may not be uniquely expressed so that final weight values are generally dependent on initial weight values and the temporal sequence of neuronal input data. The potential for multiple solutions is partially due to the use of both inhibitory and excitatory weights with the same basis function. That is, there are many values for w_i^+ and w_i^- that produce the same net weight $w_i=w_i^++w_i^-$, $i=1,...,n$. Multiple weight solutions can also exist if several complete polynomial spline spaces are used to transform the same data. The basis functions for one space may not be totally independent of those for another, and the weights corresponding to dependent basis functions could share, in varying degrees, the representation of the primary drives.

Descent Conditions

We are now in a position to derive conditions under which the learning algorithm achieves descent. Based on the prior discussion, we will assume that the magnitude of $\Delta y(t)$, $t \in T$ serves as an appropriate temporal descent function which the D-R learning algorithm must minimize with respect to the weights in order to achieve convergence. A simple approach to discovering a descent condition would be to examine how $\Delta y(t)$, in equation 2.3-1, changes with the substitution of the weights calculated for the next time-step. In particular, if we substitute the 'new' weight, calculated by equation 2.3-2, for the 'old' weight in equation 2.3-1, the projected value of $\Delta y(t)$ as a result of the weight update is given by

$$\Delta y(t)\big|_{\text{new weight}} = \Delta y(t)\left[1 + \mu \sum_{i=1}^{n} \alpha_i(t)\Delta x_i(t)\right]. \qquad (2.3\text{-}7)$$

This is a reasonable substitution because $w_i(\cdot)$ and $\Delta w_i(\cdot)$ are independent discrete variables, and hence, can be independently varied. Moreover, we have found through extensive experimental work that this equation is incredibly accurate and dependable as a basis for stability analysis and learning rate calculations. The term enclosed in the brackets determines whether to expect an increase or a decrease in $\Delta y(\cdot)$ as a result of the weight change, and can be used in an on-line setting to successfully alter the convergence characteristics of the D-R learning mechanism.

Although we did not take advantage of equation 2.3-7 for any of our work, we have used it to explain our successes and failures, and the convergence properties of D-R when used to model classical conditioning phenomenon. Note that since $\alpha_i(t) \geq 0$, $i = 1,\ldots,n$ for all $t \in \mathfrak{I}^+$, the entire fate of the D-R learning mechanism rests with the sign of the $\Delta x_i(t)$, $i = 1,\ldots,n$ terms, especially if only a few are active at any one time. For example, in the delay conditioning result of Figure 2.3-1, we have $\mu=+1$, $n=1$ and $\Delta x_1(t) \leq 0$ for all $t \in T^*$. Thus, convergence is likely for reasonable learning coefficients and a properly scaled CS. In our recent experiments, $\mu=-1$, and usually $\Delta x_i(t)>0$ when $\alpha_i(t)>0$, $i = 1,\ldots,n$ because we used relatively large receptive fields with smooth basis functions so that the positivity of these functions were strongly correlated. Thus, again, convergence was likely.

We are now in a position to propose a slight modification of the D-R learning algorithm which increases its convergence rate. This modification was not used to obtain any of the results in Chapter 4, but it does provide some additional insight into the learning mechanism itself. The discussion should be viewed as preliminary because we have not examined its impact on the ability of D-R to model classical conditioning phenomenon, and use of the modification may result in behaviors not quite consistent with the experimental evidence. The motivation for adopting this variation on the weight update formula would be to accelerate the convergence of the algorithm in complex control problems. We have observed substantial performance improvements in simple problems, but have not conducted a thorough study to determine its true effectiveness.

The modification is simple and motivated by equation 2.3-7. If we multiply the sums in equations 2.1-3 by

$$\lambda_i(t) = \begin{cases} \mu, & \Delta x_i(t) \le 0 \\ -\mu, & \Delta x_i(t) > 0, \end{cases} \tag{2.3-8}$$

for $t \in \mathfrak{I}^+$, $i \in \{1,\dots,n\}$, then it is easy to show that the equivalent expression to equation 2.3-7 is

$$\Delta y(t)\big|_{\text{new weight}} = \Delta y(t)\left[1 - \sum_{i=1}^{n} \alpha_i(t)|\Delta x_i(t)|\right]. \tag{2.3-9}$$

Thus, if the multiplier given by equation 2.3-8 is used, the convergence of the D-R learning mechanism would no longer be dependent on the sign of the changes in the conditioned stimuli. In addition, the parameter μ would no longer be required. One may keep μ for the convenience of having net weights of a certain sign, but convergence is not dependent on its value. The only requirement for descent is that the sum in equation 2.3-9 has a value between zero and two. Reasonable learning coefficients and the incorporation of an upper bound on the synaptic weights would guarantee this since there are already inherent bounds on the $\Delta x_i(t)$, $i = 1,\dots,n$ due to the compact construction of S and the partition-of-unity property of B-splines.

3. Candidate Control Structures

It is apparent, in the light of Section 2.3, why our earliest network topologies implementing D-R did not succeed, and why our latest network topologies work so well. In this chapter, we outline the progression of thought from our first experiments to the most recent, and emphasize what was important in transitioning from one network to the next. Although we attempted to generalize the character of our successful architectures, they should only be viewed as examples. There are many possible networks which could properly implement D-R, each representing different classes of control strategies or different implementations of the same strategy. We have chosen one of the more popular control strategies, that of 'feedforward' control, as the basis for this work, but it is by no means the only possible control strategy *and our final network topology is not* the only useful implementation of D-R which executes it. However, this work does provide evidence that D-R can be effectively used in control system applications.

Before proceeding, we note that in the control systems literature, 'feedforward control' does not imply the absence of feedback. Feedforward controllers often use sensory feedback since their function is to generate the 'large' control signals required to rapidly adjust to large discrepancies from a desired output. In particular, our feedforward controller uses sensory data to learn to predict the control signals required to accurately track the desired output. It can only express a true feedforward (open-loop) relationship when weights reach steady-state and learning has stopped.

This chapter is divided into three sections: Section 3.1 describes the control system architecture of our earliest experiments; Section 3.2 details the changes to this architecture that were required to achieve satisfactory results in our servo-level control experiments; and Section 3.3 discusses the final control system architecture as it was used in our successful trajectory-level control experiments.

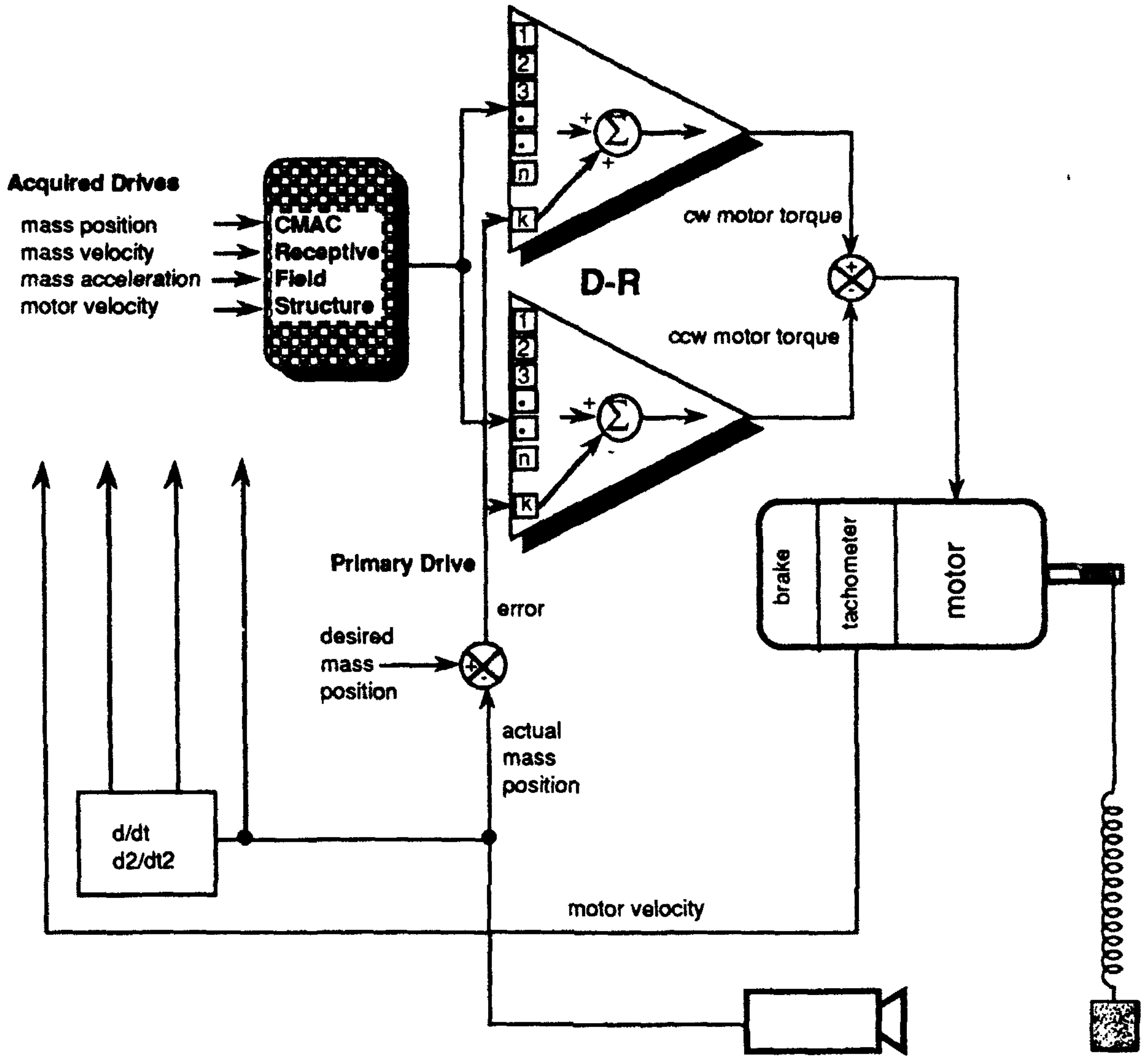

Figure 3.1-1. Configuration for the Mass/Spring Experiment.

3.1 Early Experiments

One of the earliest experiments that we performed was on the single degree-of-freedom test-bed depicted in Figure 3.1-1. The hardware consisted of a DC torque motor, a spring and a weight, connected in series with a nylon thread. The motor housing contained a tachometer to measure the speed of the motor shaft and a brake to prevent the motion of the mass until the start of an experiment. The position of the weight was determined by wrapping the thread a few times around the input to a potentiometer. The velocity and acceleration of the weight was determined through the finite differencing of the position data.

The control system for this test bed, also shown in Figure 3.1-1, consisted of a parallel connection of two D-R neurons, each capable of generating only non-negative outputs in order to be consistent with the notion that neuronal firing frequencies are non-negative. The outputs of these neurons were then subtracted in order to produce both positive and negative torque commands to the motor. The conditioned stimuli for each neuron were calculated using a CMAC receptive field structure which used motor velocity, and the position, velocity and acceleration of the weight as inputs. The error between the desired and actual weight position was used as the primary drive on the neuron generating positive torque commands. The additive inverse of this error was used as the primary drive on the other neuron. In the absence of learning, the primary drives were totally responsible for maintaining the stability of the system, and achieving some minimal level of performance.

The first experiment with this test bed required that the D-R neuron learn a constant function which would offset the gravitational force acting on the mass of the weight. The desired weight position was set equal to its initial rest position, and the brake on the motor was released to start the trial. The trial would end when the mass came to rest. This experiment appeared to yield promising results. Without learning, the weight would drop about eight inches and oscillate about its equilibrium position where the primary drive would generate enough torque to exactly offset gravity. After about 100 trials, D-R learned to generate a relatively flat torque profile, primarily due to the motor velocity signal, and prevented the weight from dropping more than an inch before coming to rest.

The second experiment with this test bed required that the D-R neuron learn a step function in the absence of the gravitational effect. After the primary drives were used to reach equilibrium, the desired weight position was increased by about three inches. Without learning, the primary drive generated an underdamped response, but eventually settled to a position three inches higher than the original equilibrium. However, after about 20 trials, D-R learned to generate a torque with the acquired drives that exactly offset the new torque due to the primary drive. Thus, since their sum (μ=+1 in these early experiments) was equal to the torque required to offset gravity, no motion toward the new goal was achieved despite the large change in the desired weight position.

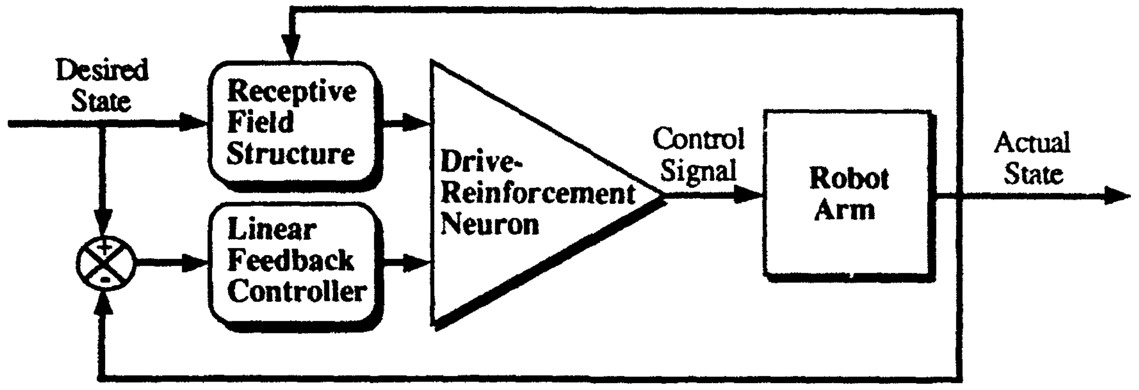
Desired State
Receptive Field Structure
Linear Feedback Controller
Drive-Reinforcement Neuron
Control Signal
Robot Arm
Actual State
+
-

Figure 3.1-2. Servo-Level Control Architecture of Early Experiments.

The control architecture of our early experiments, schematically shown in Figure 3.1-2, used either one D-R neuron, or a pair of D-R neurons as in the mass/spring experiments, to generate each control signal to the robot arm. The primary drive was always a light feedback loop capable of producing stable motions, but incapable of achieving high performance. The acquired drives were functions of sensed and/or commanded data. The objective was to enable D-R to sort the conditioned stimuli and determine which could be used to predict and reinforce the action of the primary drive, such that as the system learned, performance would improve.

The major problem with this architecture for control systems work is that it uses neurons which are responsible for both learning and motor control. As we saw in Section 2.3, the neuronal output is piecewise constant after the weights have converged, and would generally become simply constant if we used one of the 'complete' receptive field structures discussed in Section 2.2 to form the conditioned stimuli. Only in the case where the basis functions are incomplete or otherwise not representative of the primary drive will the final result be more complex, but this obscures the natural relationships and should not be depended on in practice.

The combination of CMAC receptive field structures and continuous feedback signals produced an additional problem for D-R. Out of hundreds of trajectory tracking and disturbance rejection experiments that we performed during the early part of our program, weights often diverged or produced limit cycle instabilities. This is because piecewise constant acquired drives and continuous primary drives do not work well together when used with the D-R learning mechanism. We discuss the details of this problem and its remedy in the next section.

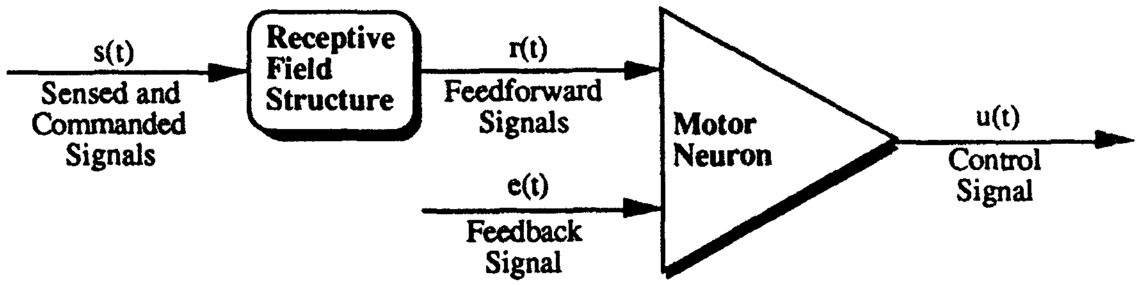

Figure 3.2-1. The Motor Neuron and Receptive Field Structure.

3.2 Progression to Current Implementation

The first alteration we made to our early control architecture was the incorporation of a separate motor neuron responsible only for the calculation of the control signals to the robot arm. The inputs to the motor neuron, as shown in Figure 3.2-1, are identical in form and number to those of the D-R neuron, and are simply weighted and summed to produce the neuronal output. They include an n-dimension feedforward signal and a scalar feedback signal. We assume here that the weights for the feedforward signals are downloaded at regular intervals (every cycle appears best) from a corresponding D-R neuron so that complex weight learning by the motor neuron is not required. We further assume that the weight for the feedback signal is equal to one, and that complex feedback calculations are not performed by the motor neuron.

As also shown in Figure 3.2-1, the motor neuron requires a receptive field structure which is identical to that of the D-R neuron. In order to be well formulated, the inputs to this structure need to occupy the same input space S as the D-R neuron, and they need to be at least dimensionally 'equivalent' to those of the D-R neuron. For example, the i^{th} component of $s(\cdot)$ for each neuron can represent the desired and actual versions, respectively, of the same signal, but must not represent radically different signals. The weighted sum of the outputs of the receptive field structure represents the total contribution of the feedforward controller to the control signal that is sent to the robot arm. As will be seen below, the control signal generated by the motor neuron can be used to actuate the robot arm, but also can be used by the D-R neuron to learn the functions required in the feedforward controller.

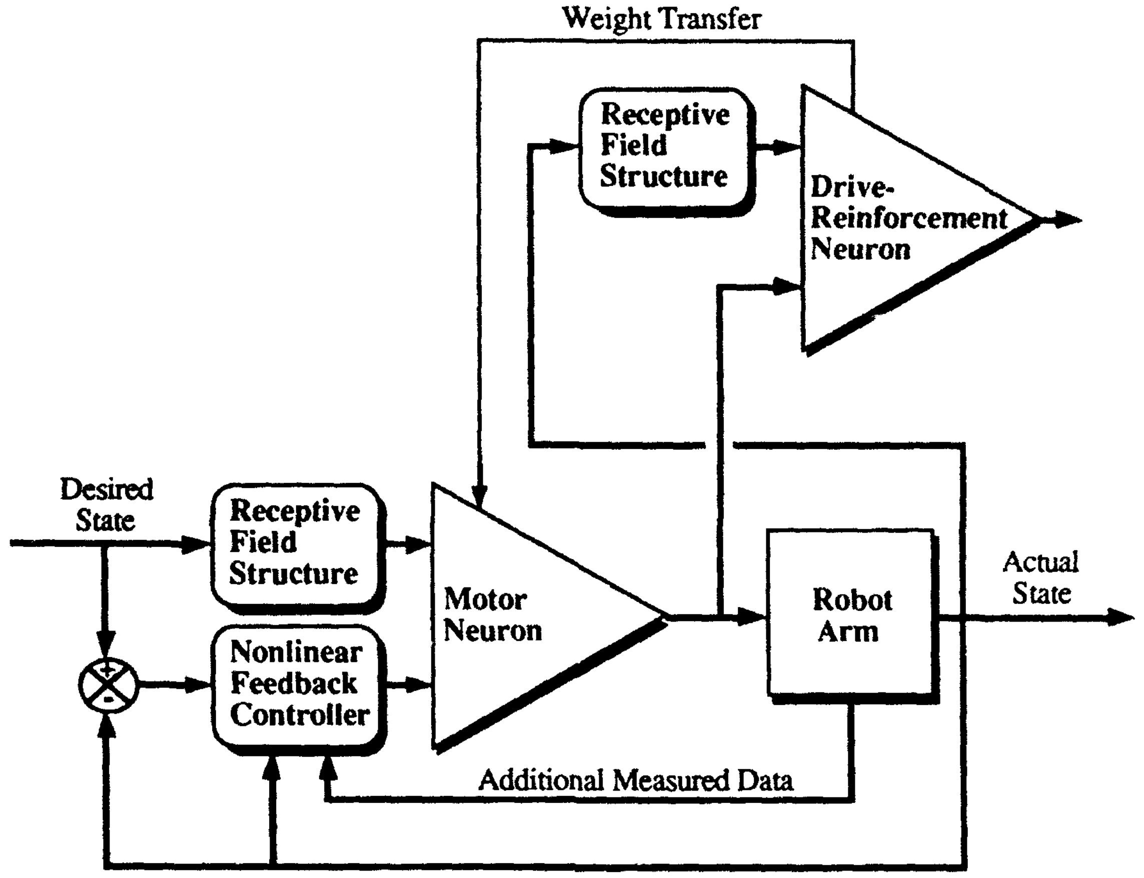
Weight Transfer
Receptive Field Structure
Drive-Reinforcement Neuron
Desired State
Receptive Field Structure
Motor Neuron
Robot Arm
Actual State
Nonlinear Feedback Controller
+
-
Additional Measured Data

Figure 3.2-2. Servo-Level Control Architecture of Intermediate Experiments.

We tried a large number of different network topologies and settled on the one illustrated in Figure 3.2-2. This topology appeared to work well, and is similar to the CMAC-based version proposed by Miller, Glanz, & Kraft (1987). Generally, in feedforward strategies, it is implied that there is a separate feedback controller, either connected in series or in parallel with the feedforward controller. The feedforward block is used to predict the large control signals required to accurately track the desired state. Since noise, disturbances and other inaccuracies corrupt this process, a feedback controller is also required to correct for any small errors in the prediction. In our case, the feedback controller is also responsible for maintaining stability with low performance expectations, and its correction to the feedforward signal enables continuous learning. Note that the output of the D-R neuron that is used in this architecture is the set of weights that is transferred to the motor neuron.

In the process of changing topology, we also abandoned CMAC receptive field structures because they often caused serious stability problems with our continuous primary drives. These problems occur during periods of continuous change in the primary drive with no change in receptive field patterns. In this case, our equation 2.3-7 provides no useful information because $\Delta x_i(t)=0$, $i = 1,...,n$ for all t in this period. In addition, we see from equations 2.3-1 and 2.3-2 that

$$\Delta y(t) = \Delta u(t) + \mu \sum_{i=1}^{n} x_i(t-1)\Delta w_i(t) - \Delta\theta(t) \qquad (3.2\text{-}1)$$

and

$$\Delta w_i(t+1) = \Delta y(t)\, \alpha_i(t),\ i = 1,\ldots,n, \qquad (3.2\text{-}2)$$

respectively. Since it is likely that $\alpha_i(\cdot)>0$ during portions of this period for some $i \in \{1,...,n\}$, it is clear that $\Delta w_i(\cdot)$ varies directly with $\Delta y(\cdot)$ which varies directly with $\Delta u(\cdot)$, all in an uncontrolled manner. Setting μ=-1 helps this situation by forcing $\Delta u(\cdot)$ to compete with the $\Delta w_i(\cdot)$, $i = 1,...,n$ in equation 3.2-1, but does not eliminate the possibility of instabilities occurring as a result of these periods. On the other hand, the use of continuous basis functions, such as our splines of order m>1, does prevent this situation since at least one basis function always changes at every time-step when the system is in motion. The utility of equation 2.3-7 is also improved due to the continuity.

The selection of the parameter μ was the only other significant change which took place in transitioning to this architecture. Setting μ=-1 had the advantage that the weights transferred from the D-R neuron would correspond directly to those required by the motor neuron for producing the feedforward term. With μ=+1, the weights would have to be inverted in sign. More importantly, according to equation 2.3-7, setting μ=-1 had stability advantages when used with our continuous basis elements because only a few of the $\Delta x_i(\cdot)=0$, $i = 1,...,n$ are active at one time and the sign of each is highly temporally correlated. Since, for computational purposes, we used relatively large receptive fields, activated by many contiguous training examples, we had to set μ=-1 to achieve stable behavior. Note, however, convergent weight behavior at every step-time can only be achieved through the use of a modification to D-R like the one that produced equation 2.3-9.

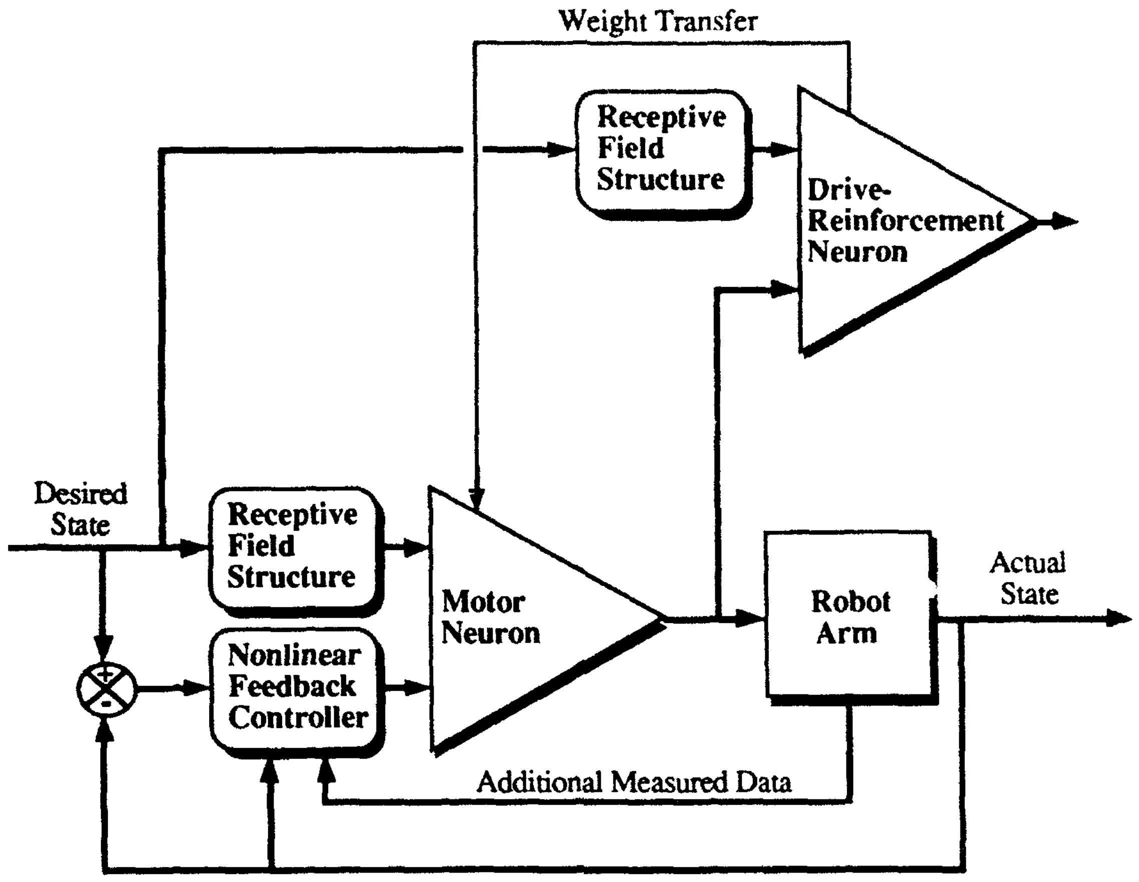
Weight Transfer
Receptive Field Structure
Drive-Reinforcement Neuron
Desired State
Receptive Field Structure
Motor Neuron
Robot Arm
Actual State
Nonlinear Feedback Controller
Additional Measured Data

Figure 3.3-1. Trajectory-Level Control Architecture of Final Experiments.

3.3 On-Line Learning Architecture

The network topology we eventually chose, shown in Figure 3.3-1, is similar to the one proposed by Kawato, Uno, Isobe, & Suzuki (1988). The only difference between this and the one of the prior section is that the desired state is used in place of the actual state as the input to the receptive field structure of the D-R neuron. This modification improves the system's performance on repeated trials of the same desired trajectory because it guarantees that one remains in the portion of input space for which learning has occurred. The modification would not be necessary, however, if either the feedback loop was very strong or the learning mechanism was very fast because in these cases the actual state will accurately track the desired state. Moreover, it does not appear to be necessary whenever one-dimensional input spaces are used to construct the conditioned stimuli. The modification was important to the success of the experiments presented in Section 4.3.

4. Applications

Early control system testing on hardware indicated that the use of complex test beds would not have allowed a thorough investigation of the underlying mechanisms for D-R learning, and consequently would have impeded our search for a proper implementation of D-R for control system applications. Hence, simple test beds were fabricated to prevent important relationships from being obscured during experimentation, and to enable us to obtain a better understanding of the resulting system behavior. The results presented in this chapter are a sampling of our successes with D-R, and were obtained using a two degree-of-freedom manipulator test-bed. Simulation results are also presented, and compared with those obtained from hardware.

Our performance evaluations of D-R based controllers always began with detailed computer simulations of the actual robot arm and its control system. Using identical software structures for the simulation and the real-time code allowed control software to be directly transferred from the computer simulations to the physical control computers without alteration, and had the effect of promoting safer, more reliable system operation while reducing the time required for software integration on the manipulator test-bed. In addition, since evaluations of D-R networks could be performed faster and at less risk in simulation than on the hardware, many more tests could be performed than was possible in the same amount of time using the test-bed alone. In this way the test bed could also be reserved for evaluating effects only found in a hardware implementation.

This chapter is divided into three sections: Section 4.1 describes mechanical, electrical and software aspects of our planar arm test-bed; Section 4.2 details the results of our intermediate servo-level control experiments; and Section 4.3 details the results of the trajectory-level control experiments which utilized our latest control system architecture for on-line learning.

4.1 Planar Arm Test-Bed

Our planar arm is a two degree-of-freedom manipulator constructed in a horizontal plane from the upper arm and elbow joints of the anthropomorphic arm test-bed shown in Figure 4.1-1. Each joint has a maximum speed of approximately 500 deg/s, and is driven by a brushless pulse-width modulated DC motor through a 59:1 cycloidal cam gear reducer. They are equipped with resolvers on the motor shafts for sensing angular position, and limit switches used for generating emergency stops near physical hardstops and for the calibration of angular position measurements. The test bed is also equipped with an optical end-point sensing system that measures the Cartesian position of the tip of the robot arm.

The computer control system in this test bed consists of a standard nineteen inch electronics rack with motor controllers, amplifiers, signal conditioning hardware, safety system hardware, and a VME backplane. A single 68030 microprocessor-based VME board with a floating point coprocessor performs all of the real-time control and safety functions. A second 68020-based processor board is used to collect and store performance data so that it can be uploaded to the host Sun workstation for subsequent analysis and plotting. The analog-to-digital and digital-to-analog conversions are handled on a separate board in the VME backplane, under the direct control of the 68030 processor board.

Sun workstations are used to develop our control system software in portable C language modules that can be compiled and run both in our simulations on the Sun as well as in the real-time controller on the 68000 series microprocessors. The Sun workstation communicates with the VxWorks operating system running on the two processors in the VME backplane using a standard ethernet interface. Thus, our software is compiled on the Sun, and then conveniently and quickly downloaded to the two control system processors. The ethernet link between the Sun workstations and the control system processors provides the added capability of directly transferring collected data from the control system to the Sun or any other computer on our ethernet network. We can thus take advantage of commercially available software packages for design and analysis such as MATLAB and for graphing such as our X-windows based plotting program called xgraph.

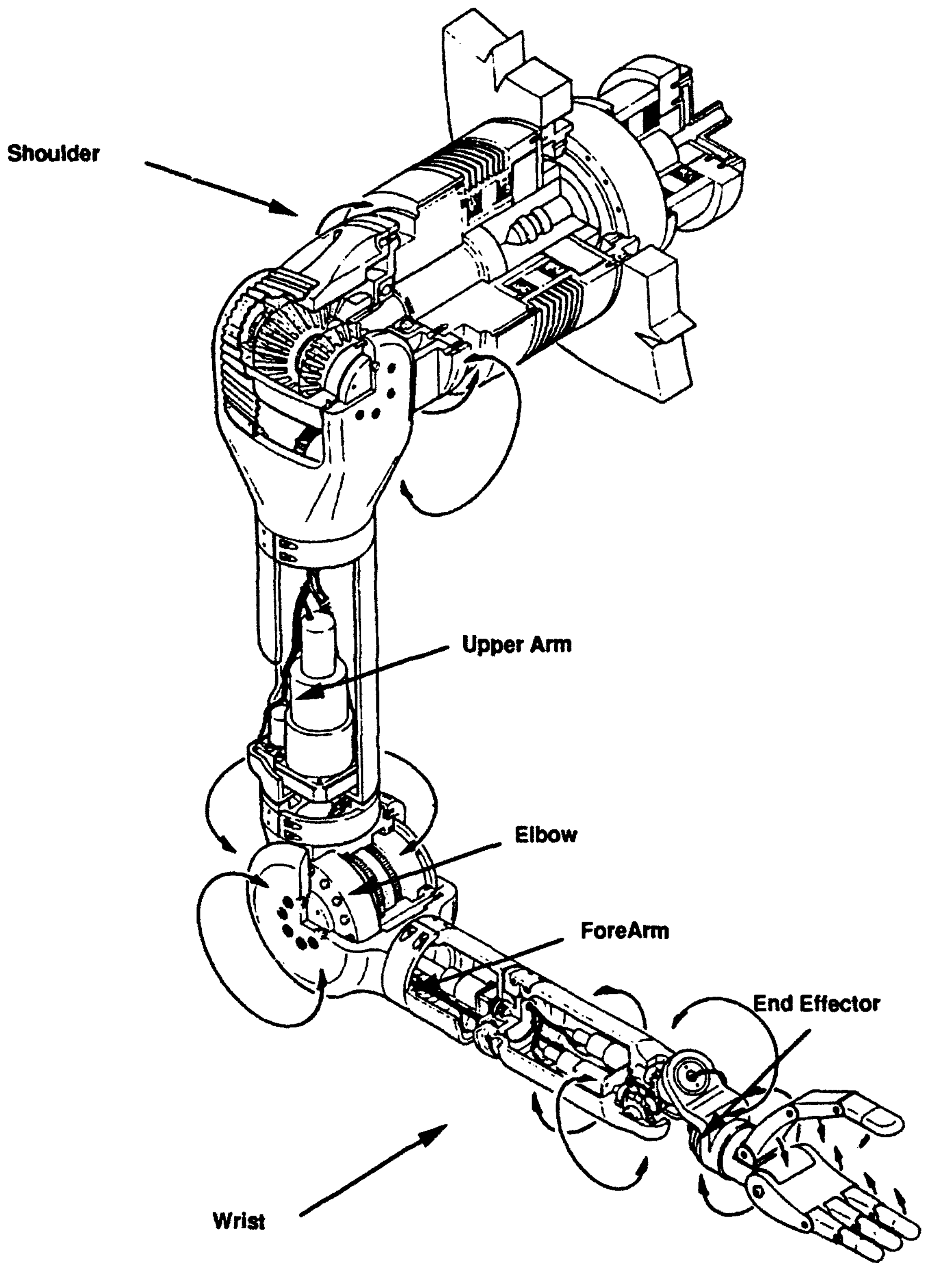

Figure 4.1-1. Geometry of the Complete Anthropomorphic Arm.

4.2 Servo-Level Control Experiments

The D-R implementation shown in Figure 3.2-2 was used in this study of the on-line learning of servo-level control. With it, the D-R neuron learns to approximate the inverse dynamics of the planar robot arm. By transferring the weights learned in each control cycle to the motor neuron, the motor neuron predicts the control signals required to accurately track the desired state. In this case, the control signals are voltage inputs to the joint actuators, the desired state contains commanded joint angles and joint velocities, and the 'system' is the planar robot arm with its actuation devices. Since the actuators for this arm have sufficient internal feedback to essentially decouple the interaction between the joints, one-dimensional piecewise-linear B-splines were used to develop the receptive field structure. Per the discussion in Section 2.3, the B-spline closest to the origin was eliminated in order to force the output of the D-R neuron to zero.

The feedback control law for the servo-level control experiments was simply a constant gain on each of the joint position errors. The gains required to produce stable motions in the absence of learning were simple to establish. This feedback law works because the actuators look like velocity control devices so that the resultant joint velocities are roughly proportional to the actuator input. Since the proportionality factor may be nonlinearly dependent on the state of the system, the task of the D-R learning mechanism is to compute it as a function of the state of the system.

This implementation of D-R was first studied in simulation and then transferred to the planar robot arm test-bed in order to validate the concept on hardware. In particular, a series of trajectory-tracking experiments were conducted to test how well the D-R neurons could learn to accurately follow commanded data in the presence of real sensor noise, unmodeled actuator and arm dynamics, sample-data effects, computational delays and parameter errors. We found that D-R was successful with hardware utilizing this implementation, and these results were found to be in close agreement with those predicted from simulation studies.

For reference, a sample of simulation time-histories are shown in Figures 4.2-1 to 4.2-5, and some of the corresponding hardware results are shown in Figures 4.2-6 to 4.2-9. Positions are expressed in degrees, velocities in degrees-per-second and time in seconds.

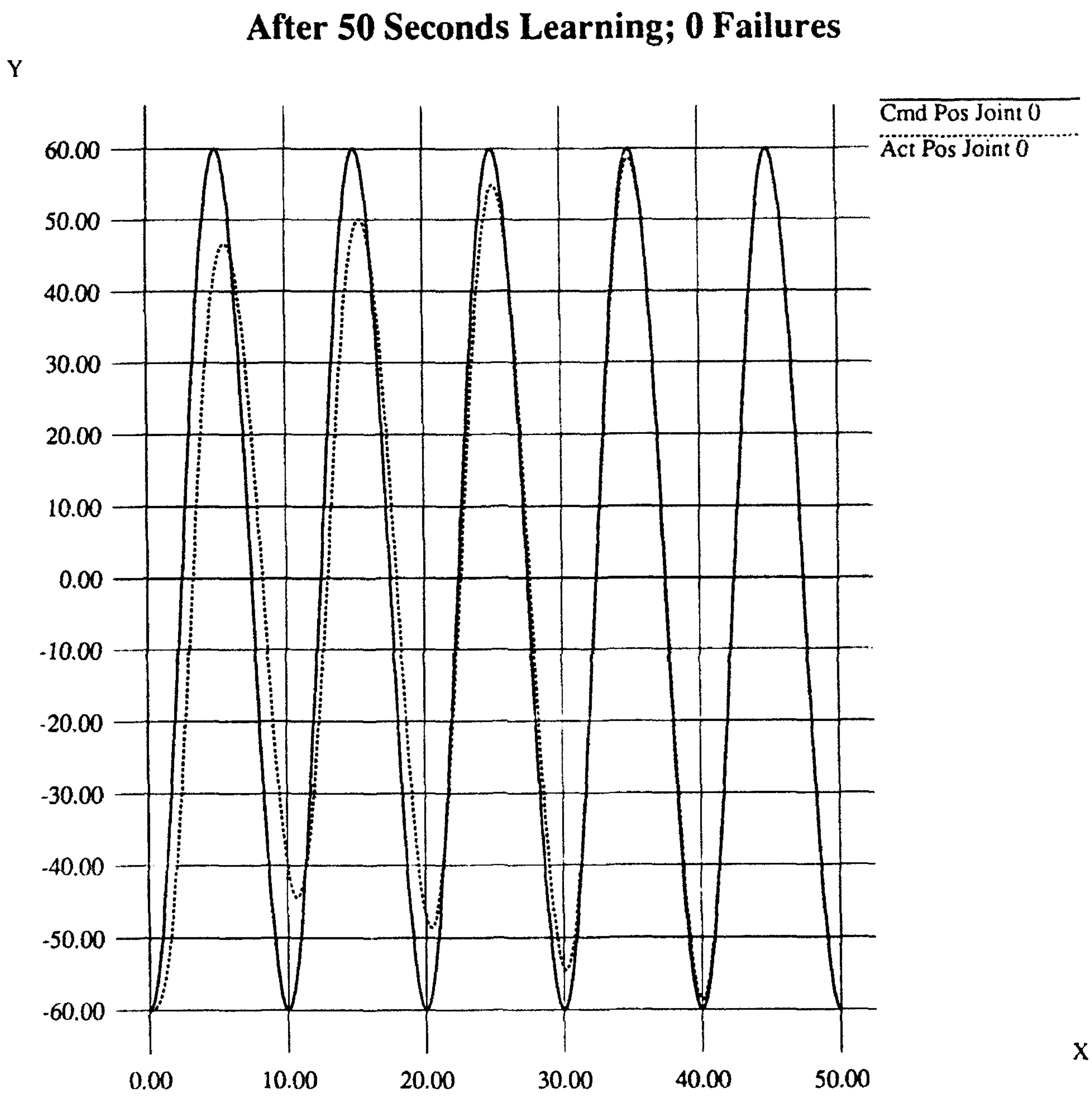

Figure 4.2-1a. Commanded and Sensed Upper Arm Positions.

After 50 Seconds Learning; 0 Failures

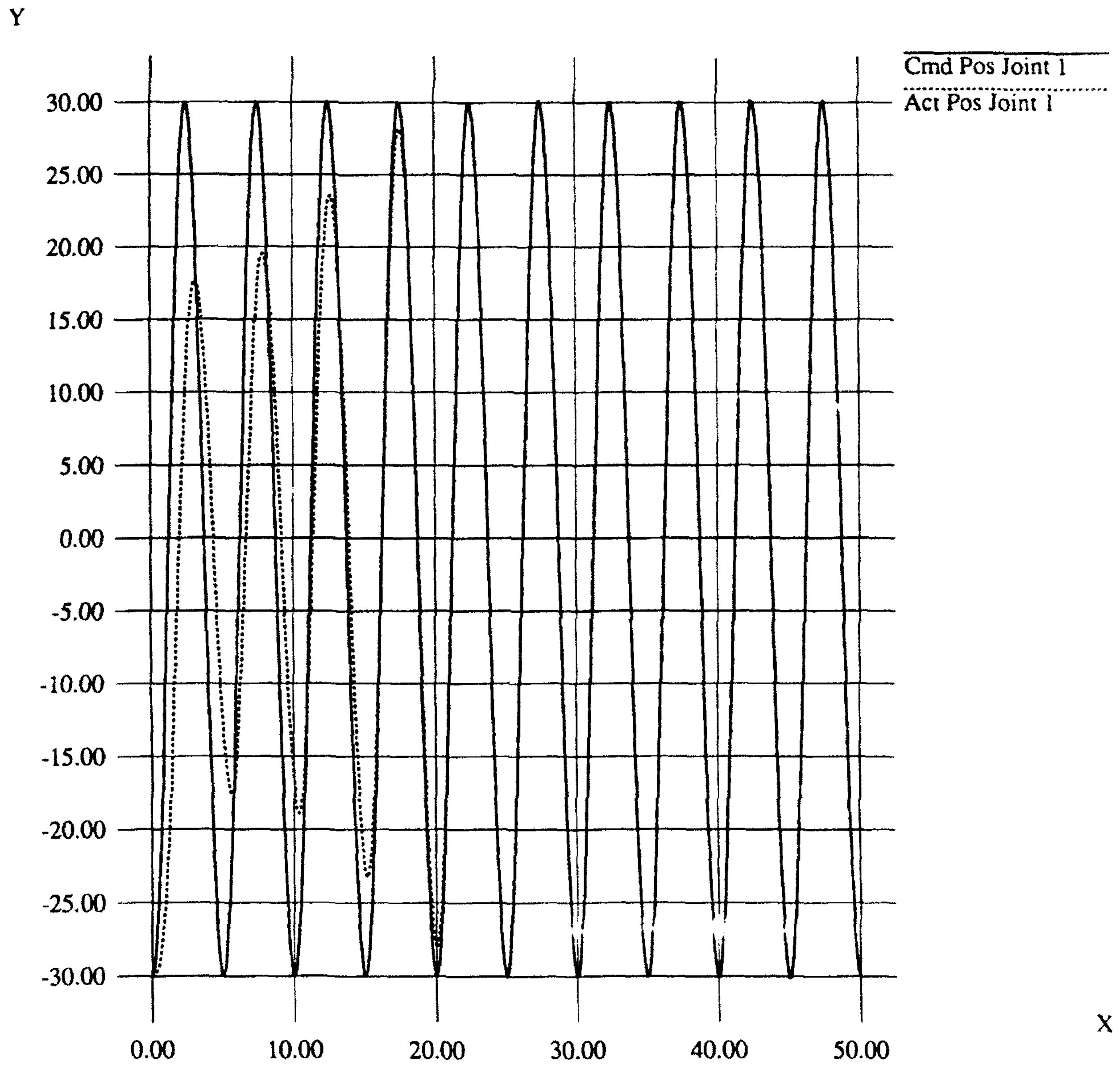

Figure 4.2-1b. Commanded and Sensed Elbow Positions.

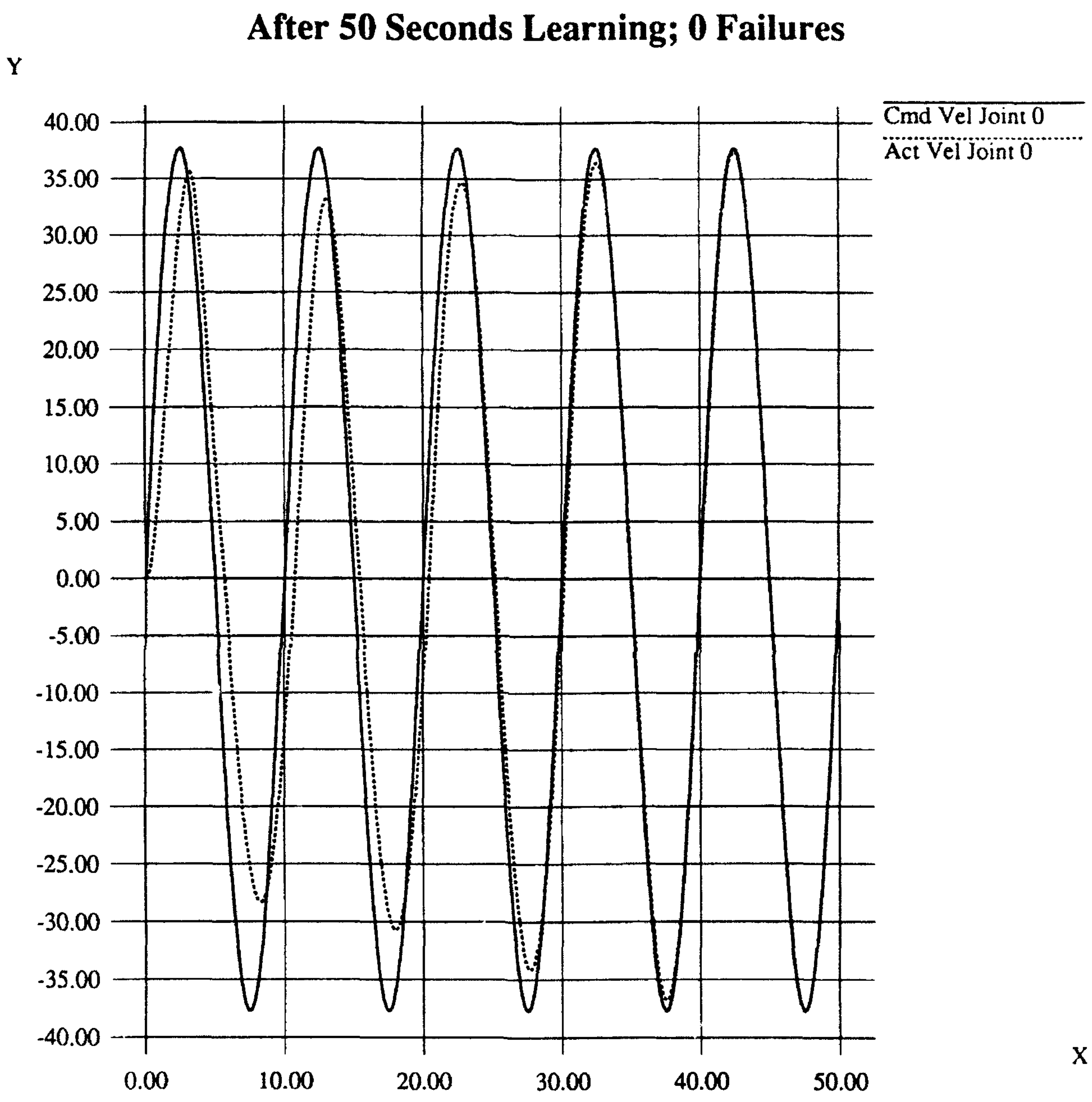

Figure 4.2-2a. Commanded and Sensed Upper Arm Velocities.

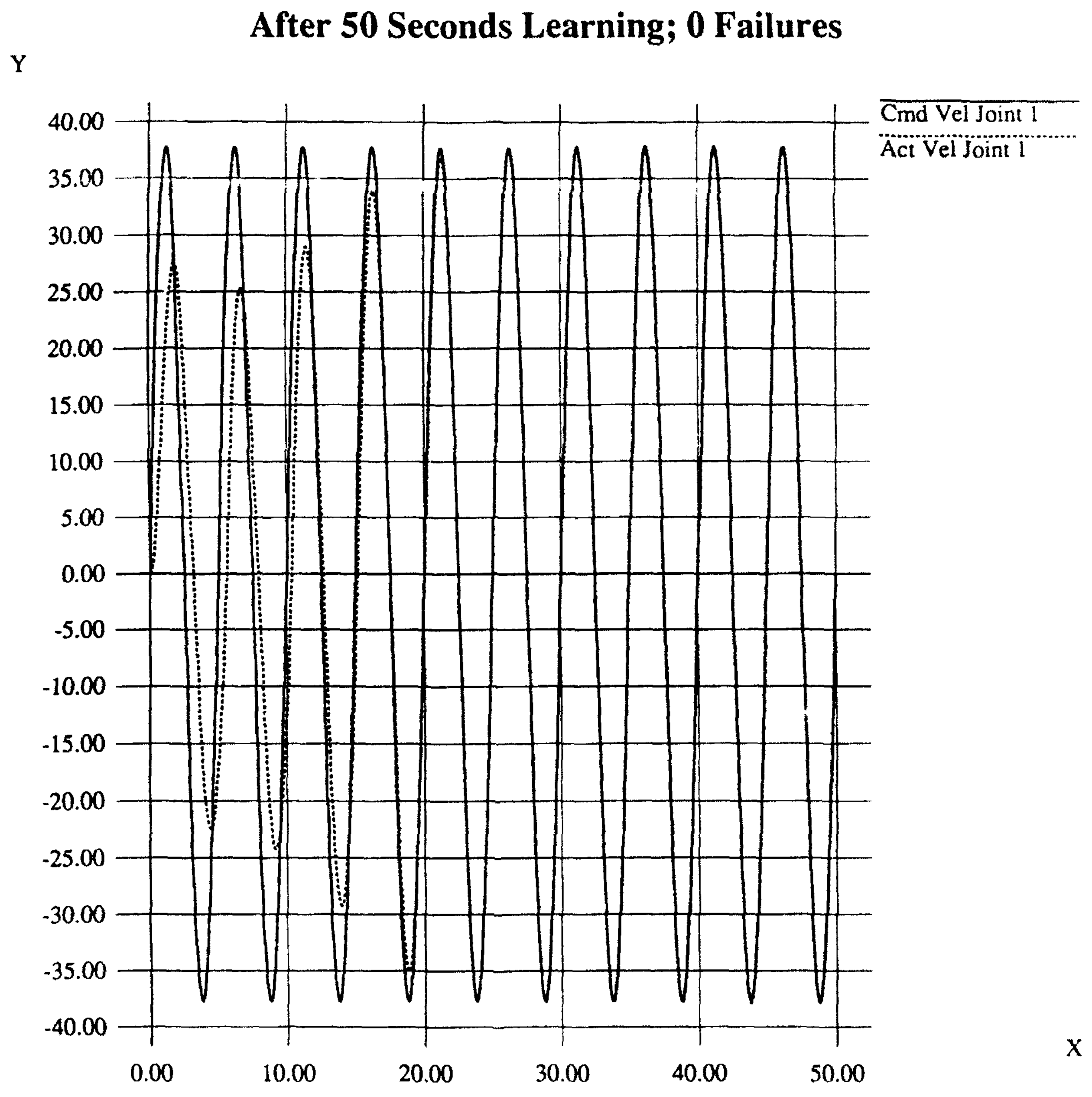

Figure 4.2-2b. Commanded and Sensed Elbow Velocities.

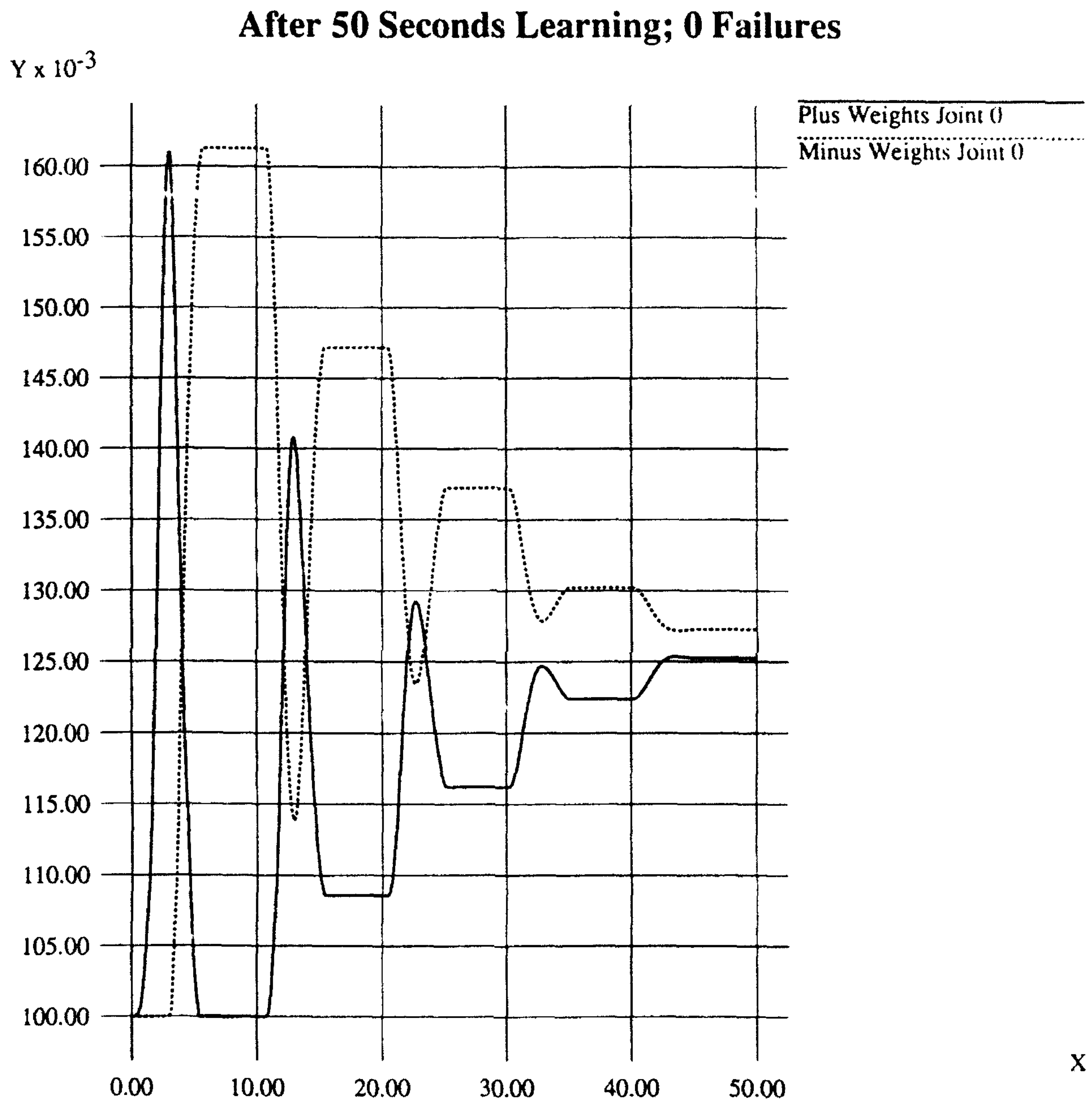

Figure 4.2-3a. Upper Arm Position Weights.

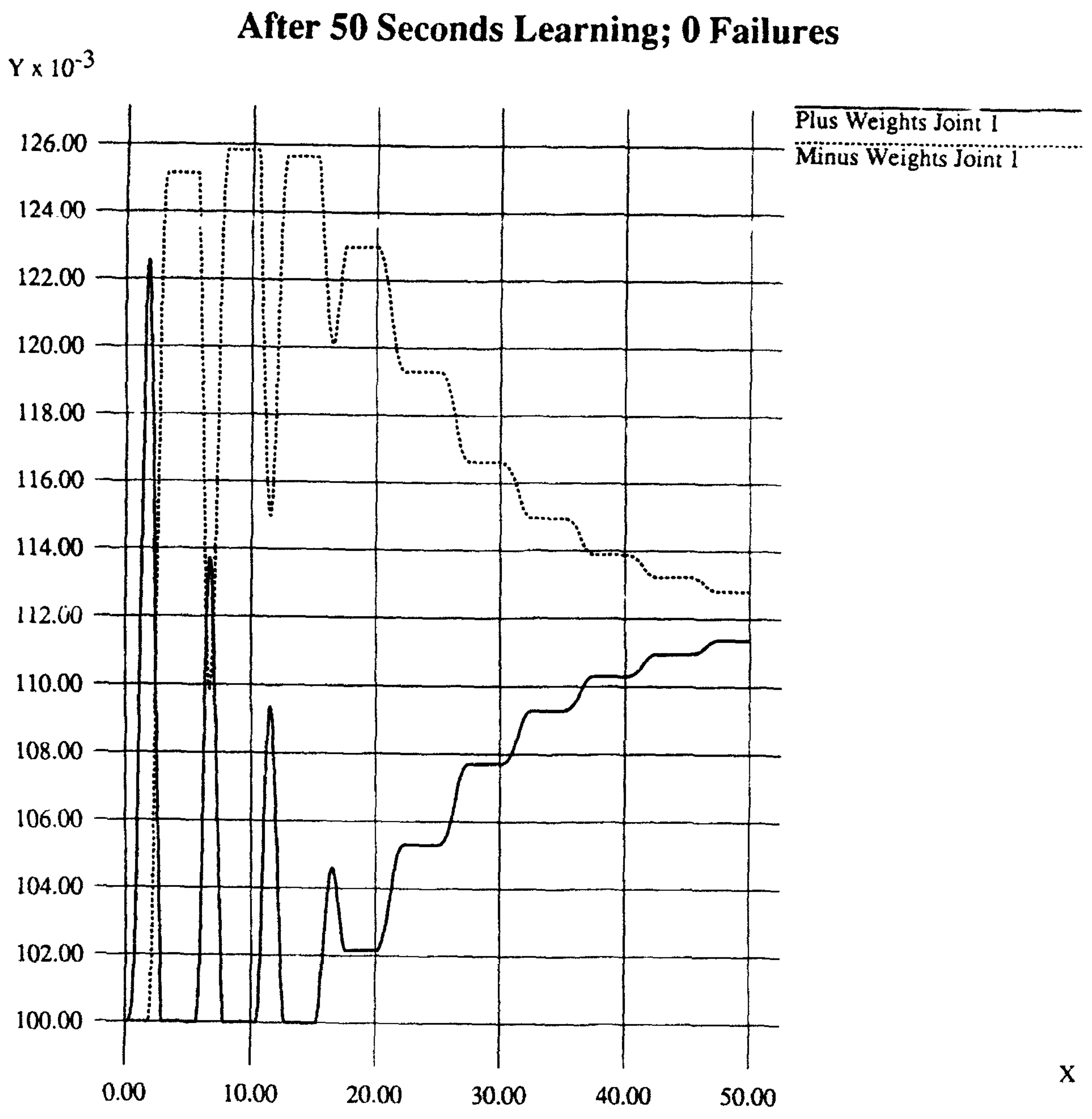

Figure 4.2-3b. Elbow Position Weights.

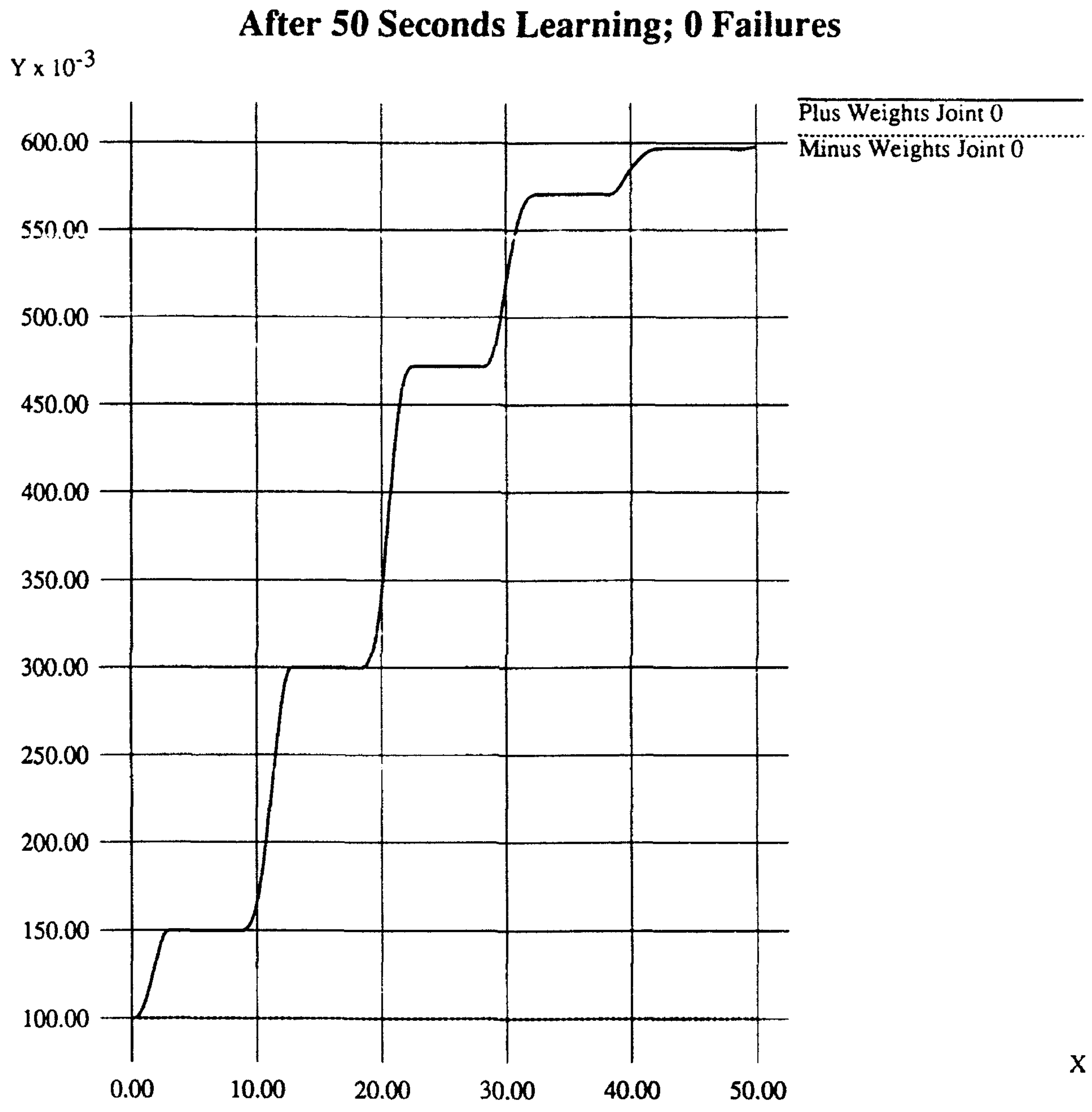

Figure 4.2-4a. Upper Arm Velocity Weights.

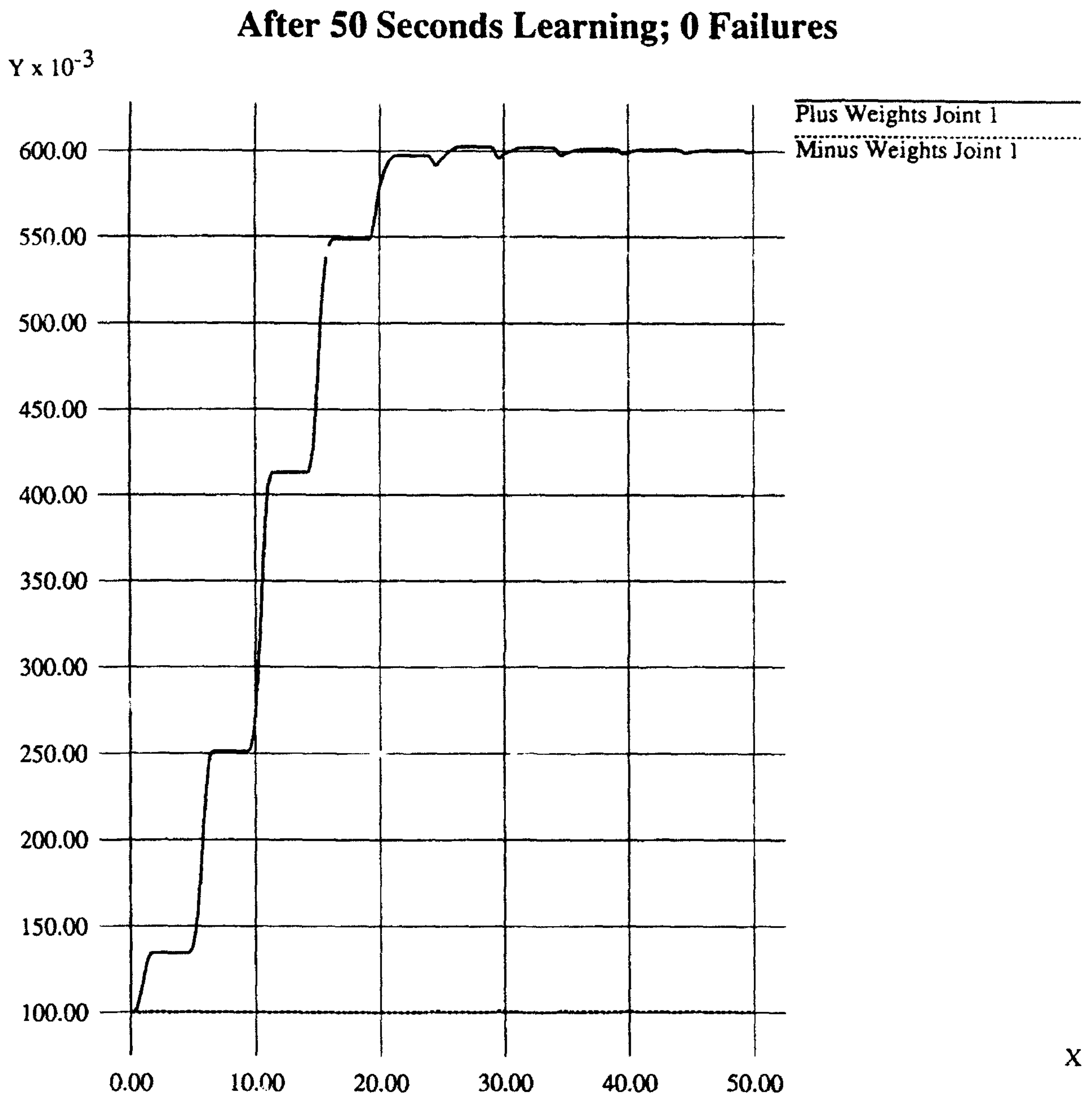

Figure 4.2-4b. Elbow Velocity Weights.

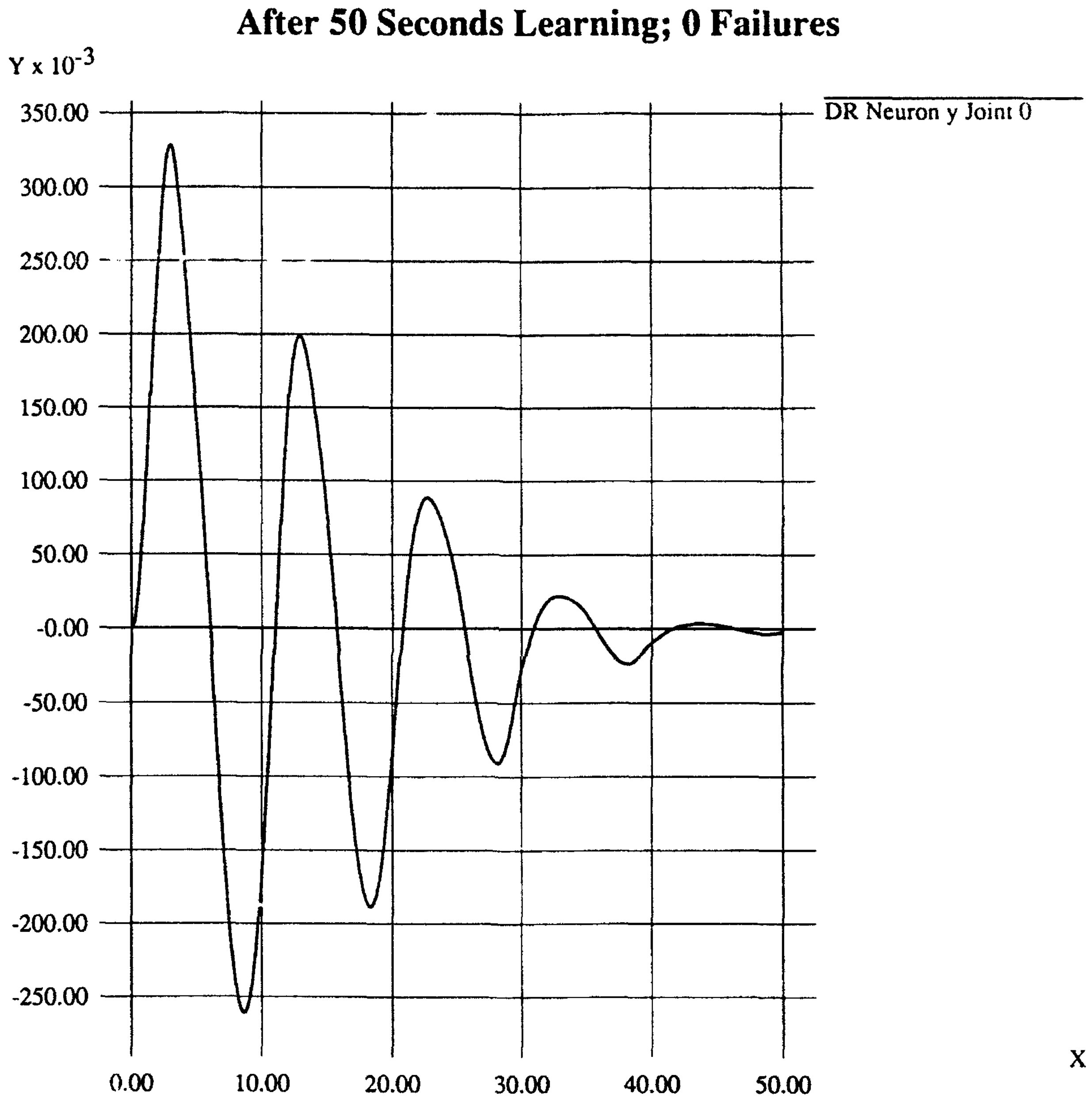

Figure 4.2-5a. Output of the D-R Neuron for the Upper Arm Joint.

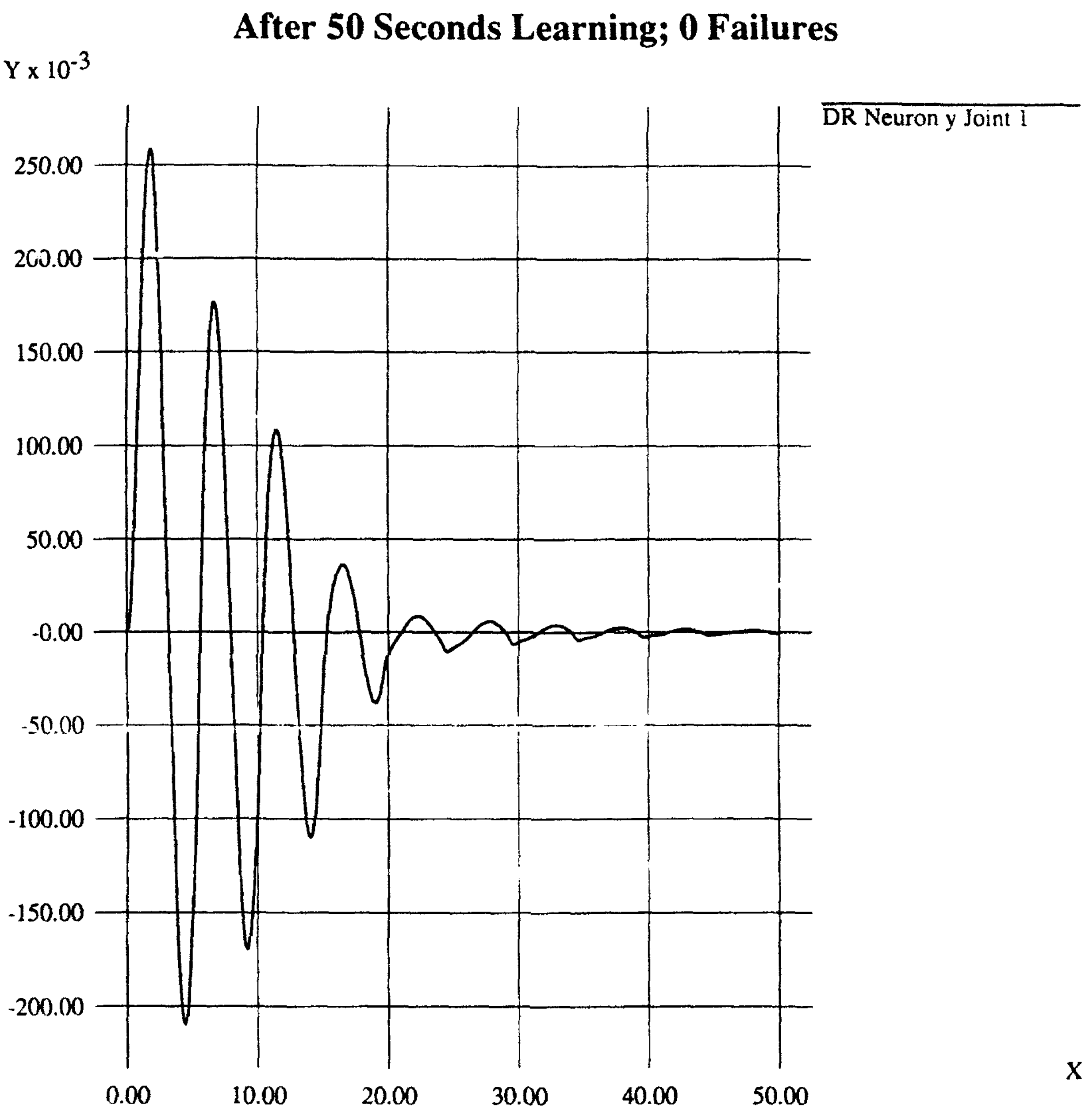

Figure 4.2-5b. Output of the D-R Neuron for the Elbow Joint.

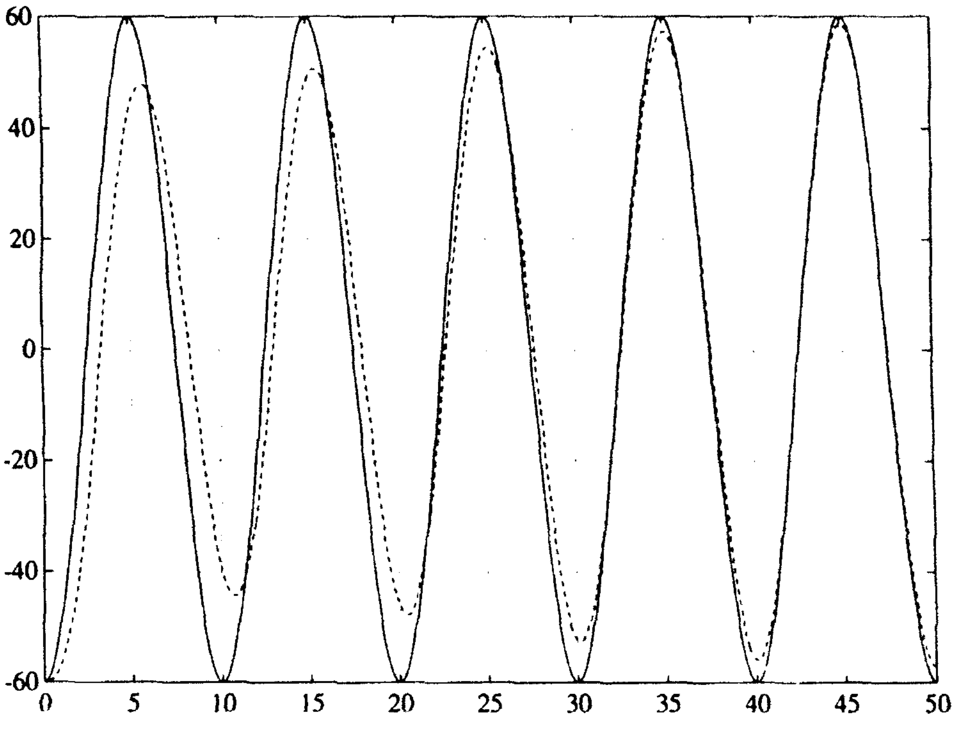

Figure 4.2-6a. Commanded and Sensed Upper Arm Positions.

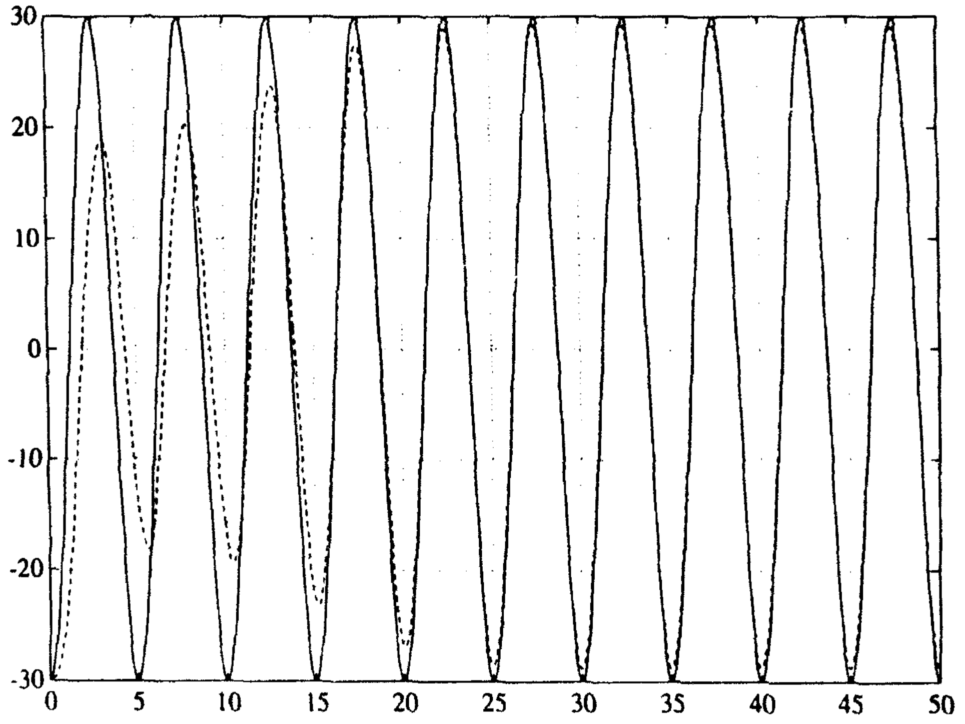

Figure 4.2-6b. Commanded and Sensed Elbow Positions.

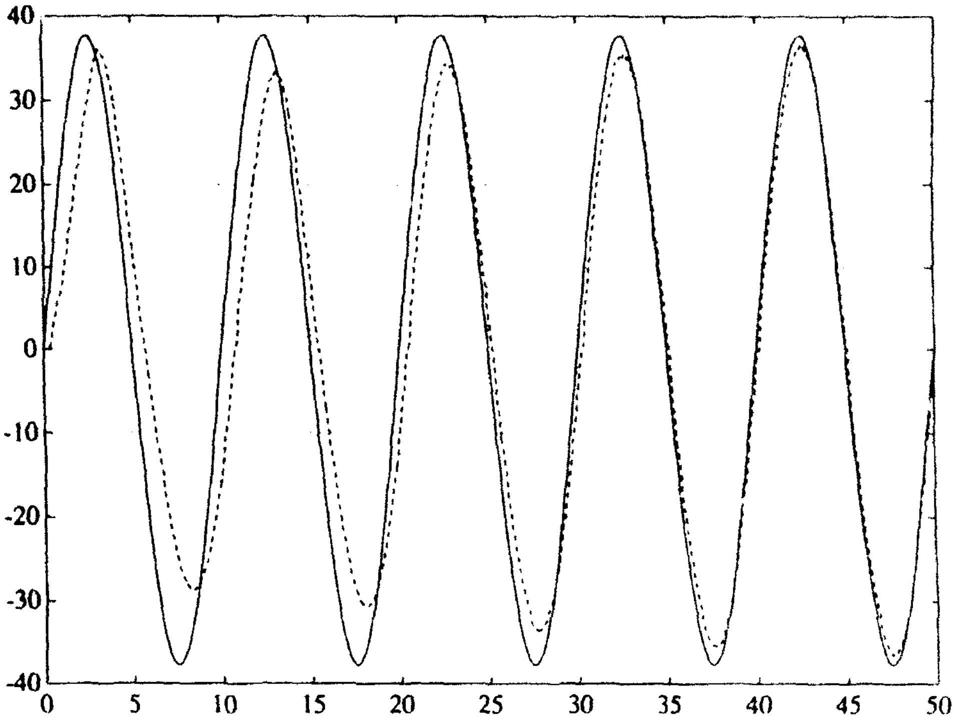

Figure 4.2-7a. Commanded and Sensed Upper Arm Velocities.

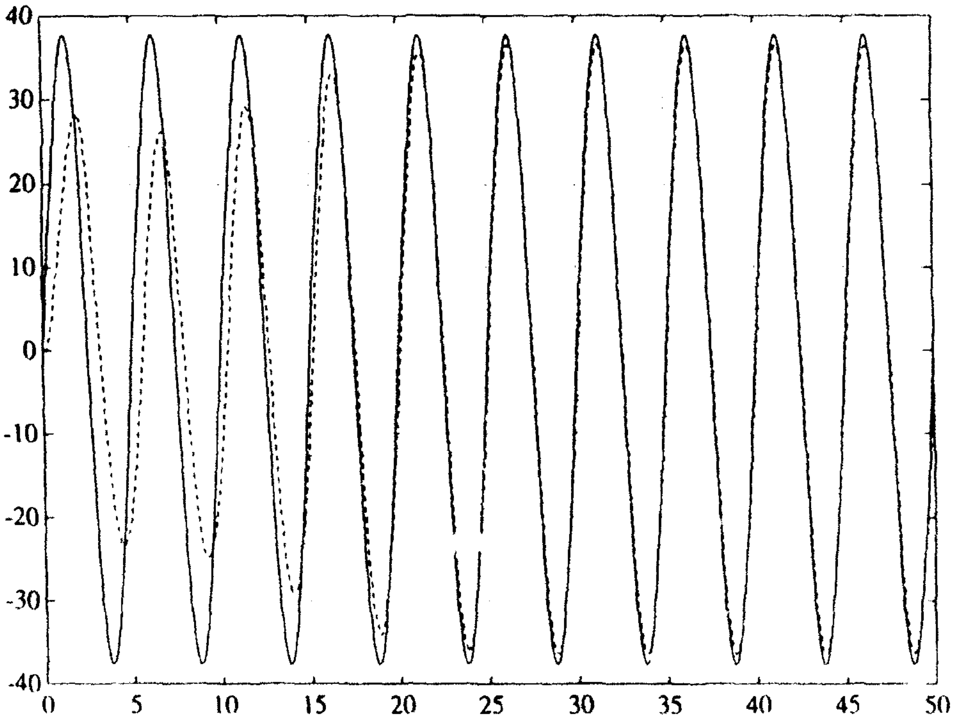

Figure 4.2-7b. Commanded and Sensed Elbow Velocities.

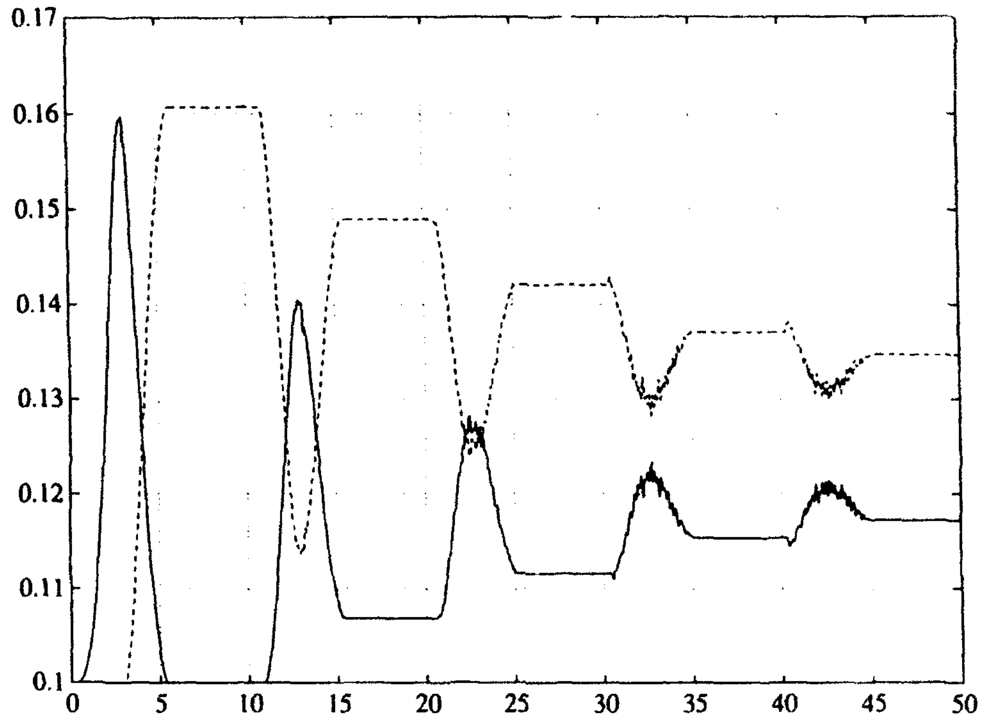

Figure 4.2-8a. Upper Arm Position Weights.

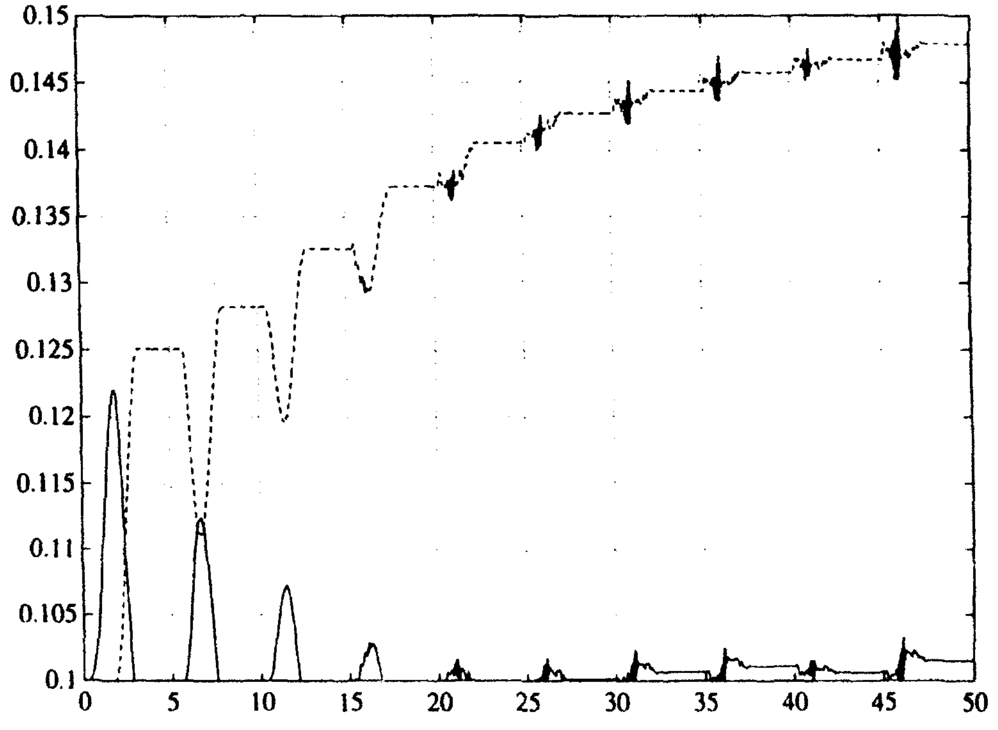

Figure 4.2-8b. Elbow Position Weights.

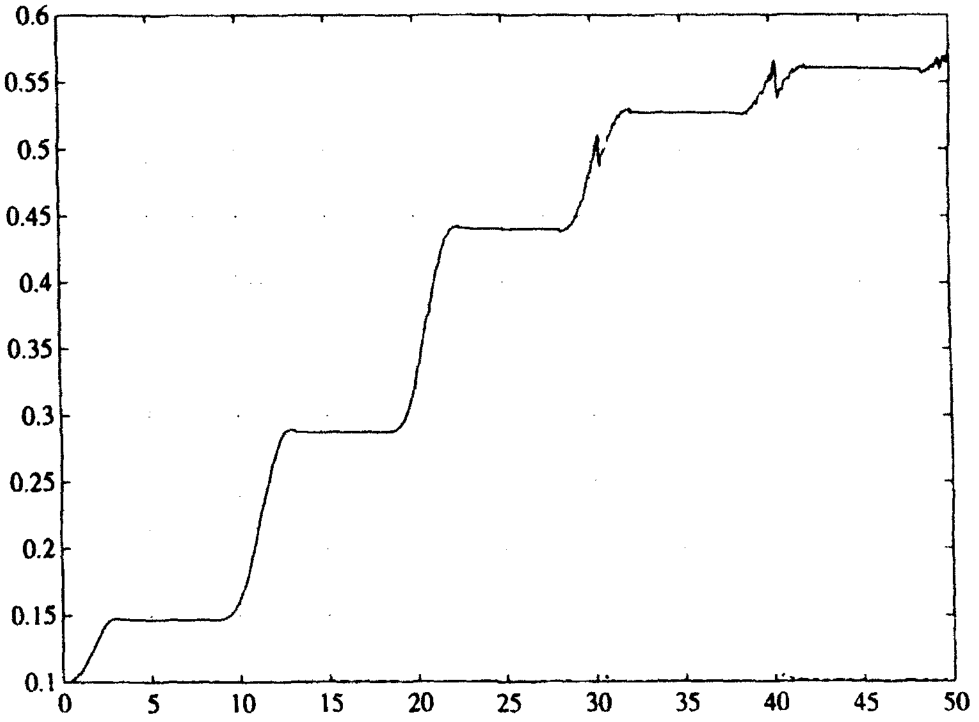

Figure 4.2-9a. Upper Arm Velocity Weights.

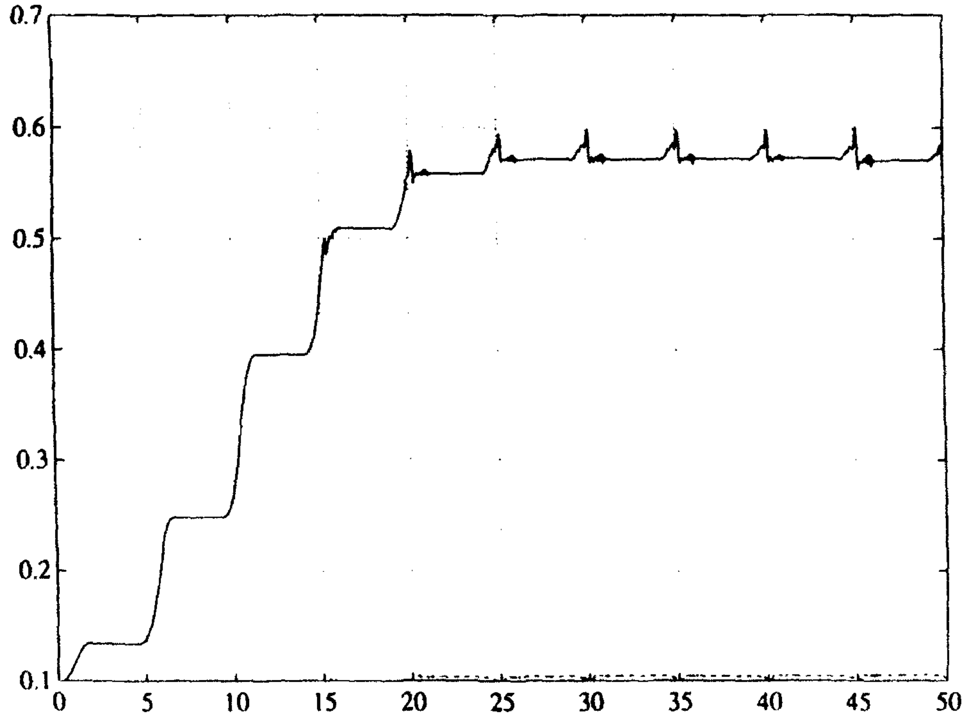

Figure 4.2-9b. Elbow Velocity Weights.

4.3 Trajectory-Level Control Experiments

The D-R implemen.ation shown in Figure 3.3-1 was used in this study of the on-line learning of trajectory control. With it, the D-R neuron learns to approximate the inverse kinematics of the robot arm under control. Using the weights learned in each control cycle by the D-R neuron, the motor neuron can predict the control signals required to accurately track the desired state. In this case, the control signals are 'desired' manipulator joint angles, the desired state is the commanded Cartesian position of the tip of the robot arm, and the 'system' consists of the planar robot arm and its servo-level control system. To accommodate the dimensionality and coupling of the components of the desired state, two-dimensional tensor-product B-splines were used to develop the receptive field structure. To avoid the elimination of basis elements, equation 2.3-6 was used to calculate the neuronal threshold required to force the output of the D-R neuron to zero.

The nonlinear feedback control law for the trajectory-level control experiments had the form of a two-dimensional matrix-vector product, one component of which was used in each motor neuron. The matrix is an approximation to the inverse Jacobian of the kinematic plant, and is computed with only real-time data using a standard rank-one update formula similar to that found in Strang (1986). The vector is a simple linear feedback element consisting of the proportional and integral feedback of the Cartesian position errors. It was easy to find feedback gains which produced stable motions, in the absence of learning, using this controller.

This implementation of D-R was first studied in simulation and then transferred to the planar robot arm test-bed in order to test the concept on hardware. We found that D-R was successful with the hardware utilizing this implementation, and these results were found to be in close agreement with those predicted from simulation studies.

For reference, a sample of simulation time-histories are shown in Figures 4.3-1 to 4.3-3, and time-histories captured from the hardware experiments are shown in Figures 4.3-4 to 4.3-8. Here, positions are expressed in inches and time in seconds.

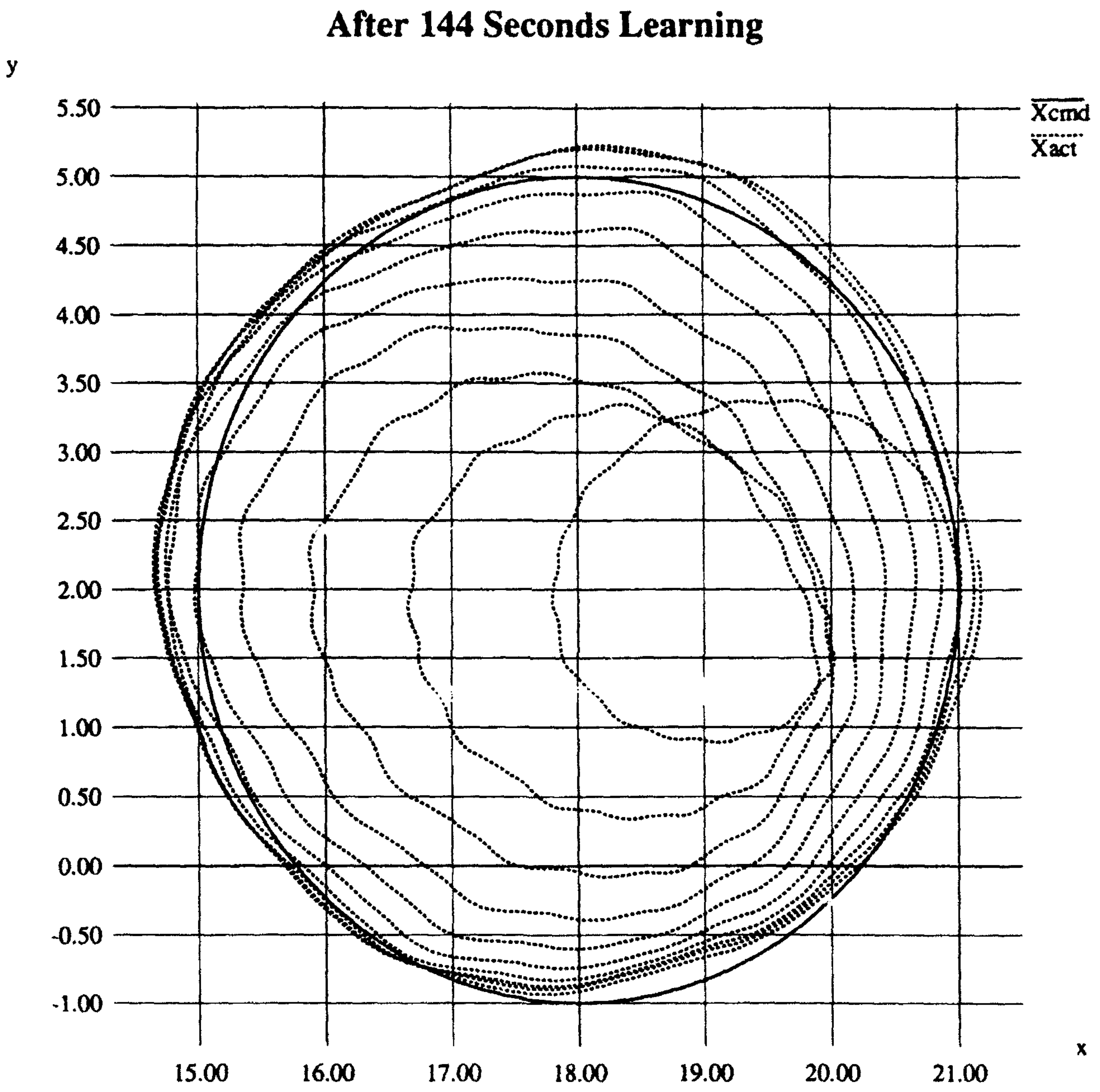

Figure 4.3-1a. Commanded and Sensed X-Y Positions With Learning.

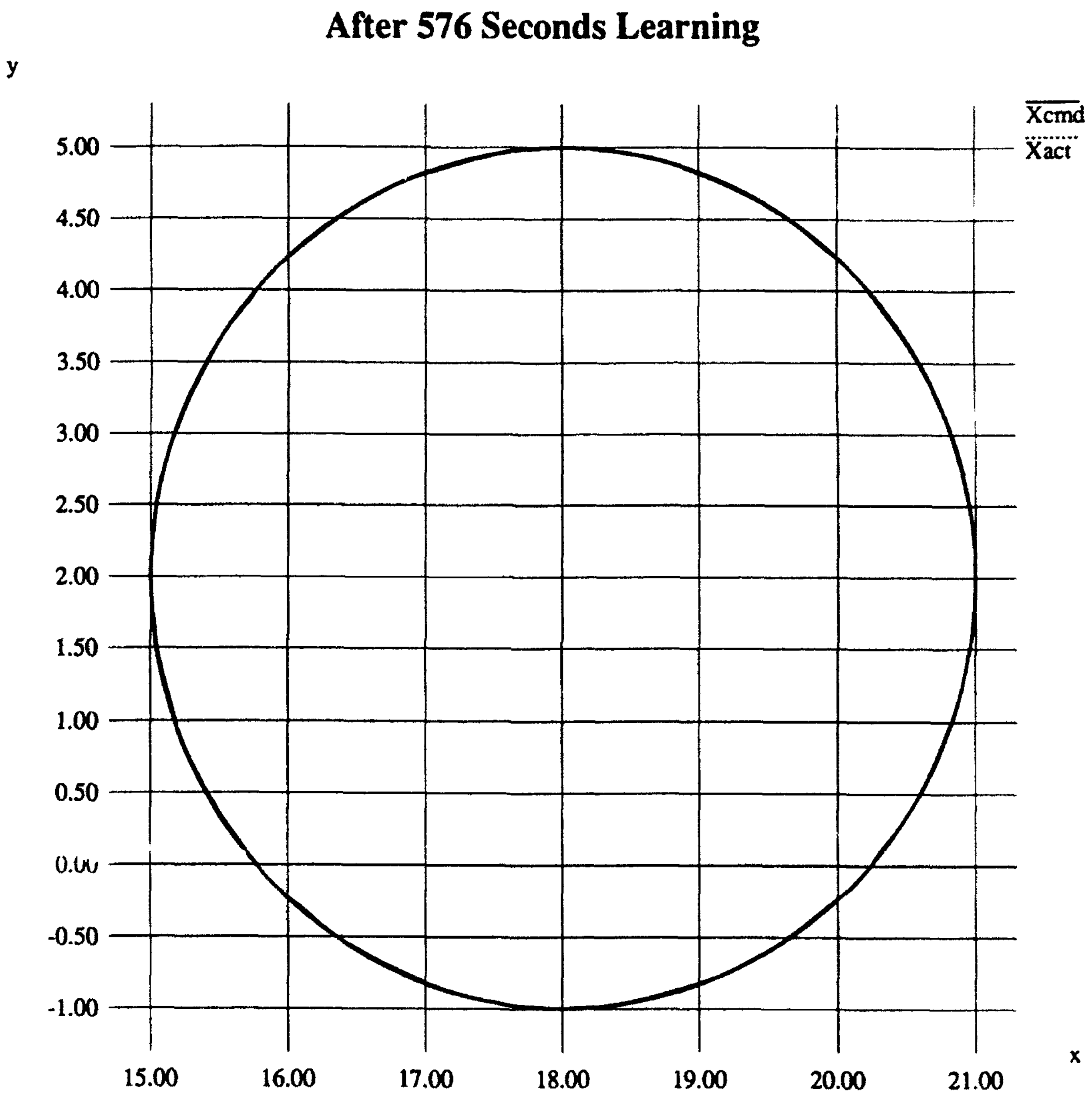

Figure 4.3-1b. Commanded and Sensed X-Y Positions With Continued Learning.

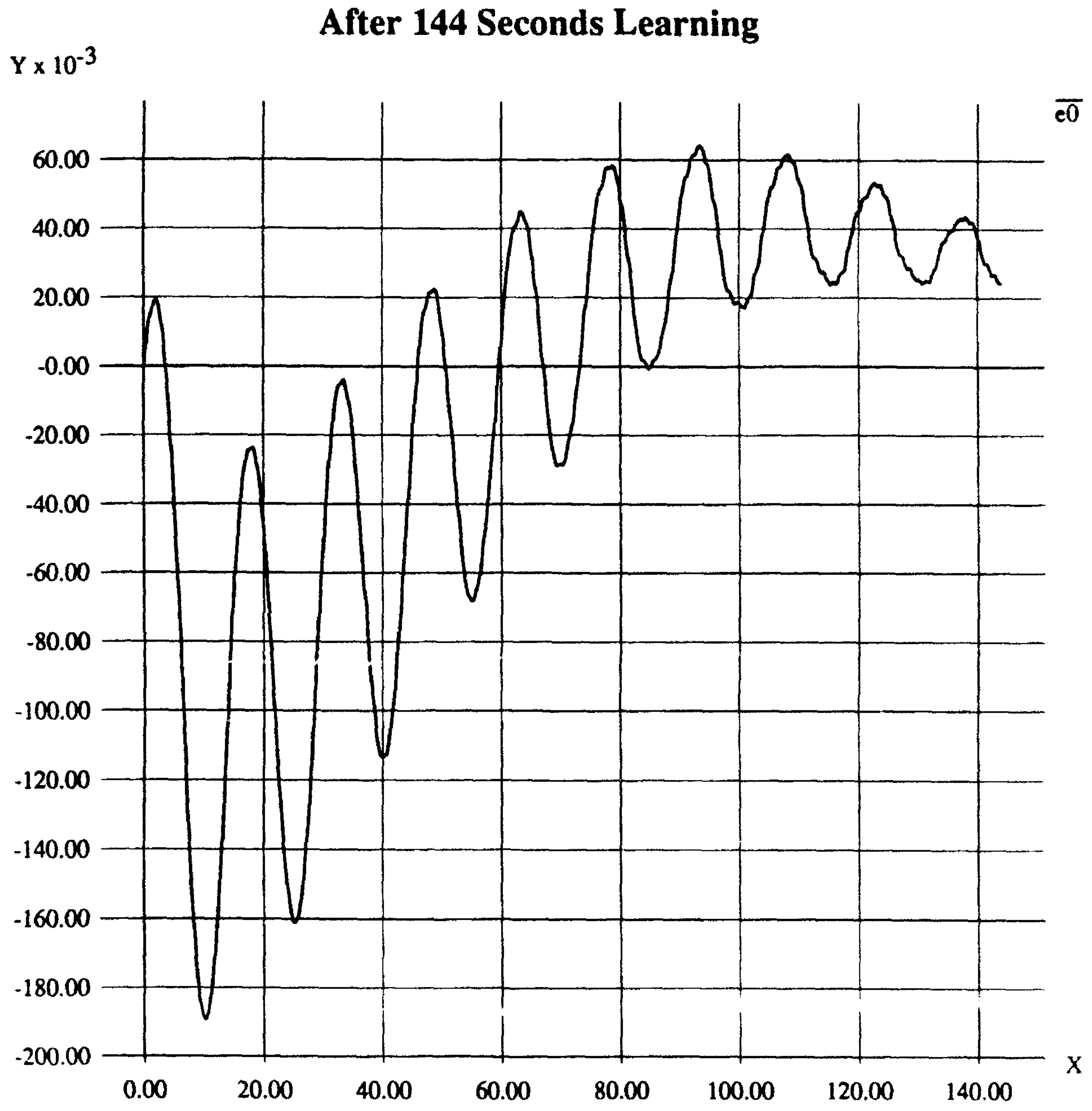

Figure 4.3-2a. Output of the D-R Neuron for the Upper Arm Joint.
This is the first 144-second interval of learning.

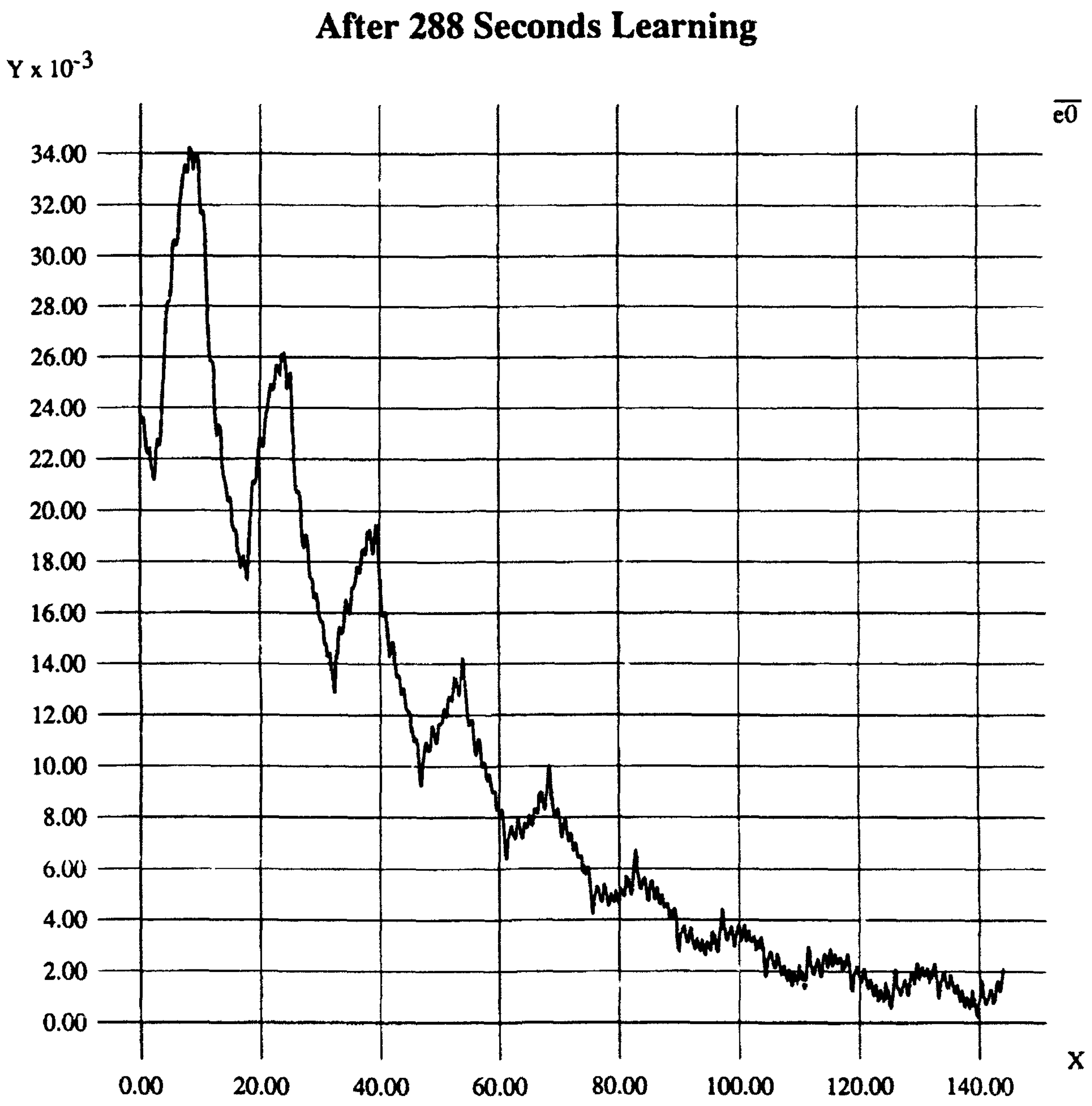

Figure 4.3-2b. Output of the D-R Neuron for the Upper Arm Joint (Continued). This is the second 144-second interval of learning.

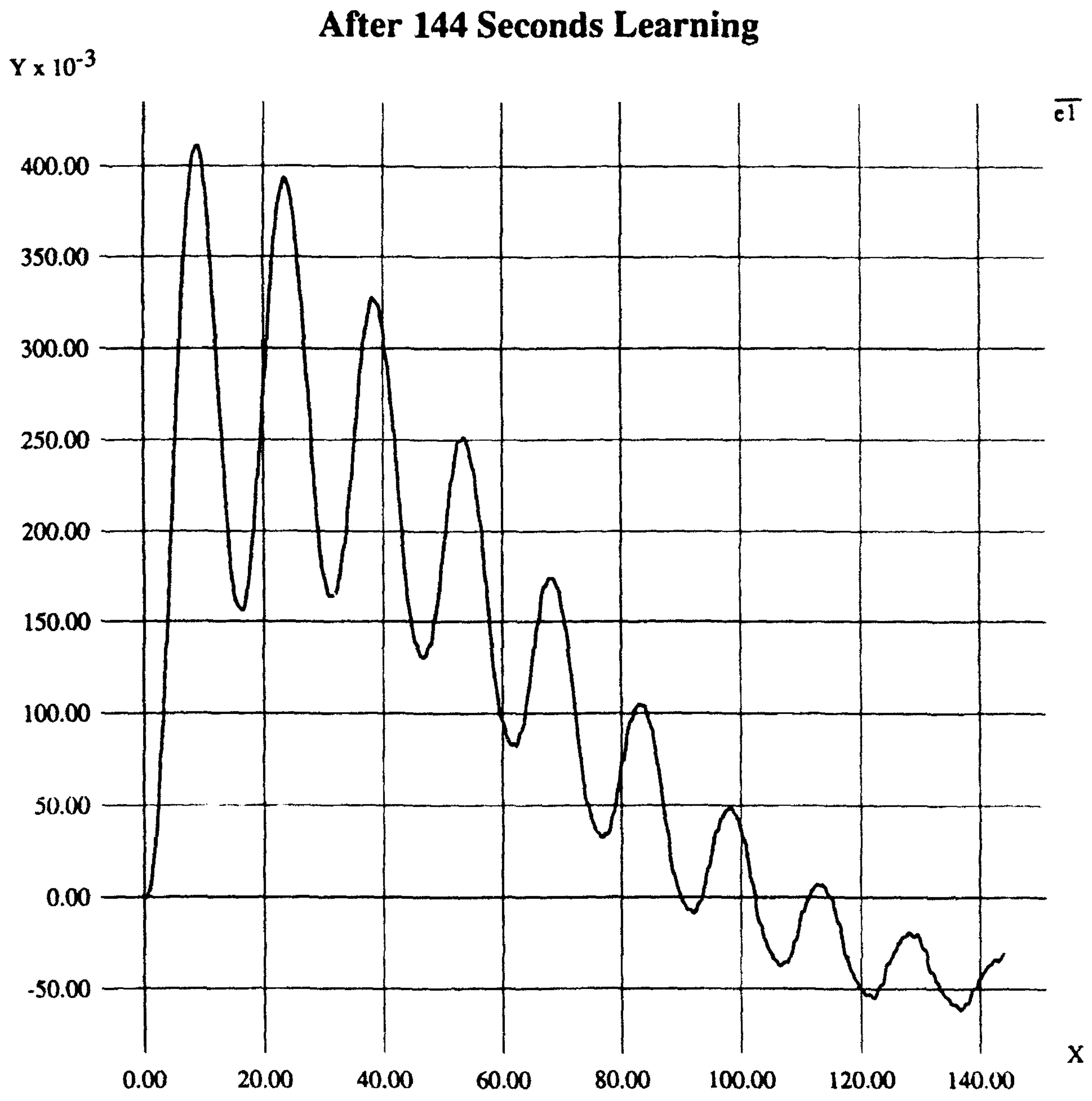

Figure 4.3-2c. Output of the D-R Neuron for the Elbow Joint.
This is the first 144-second interval of learning.

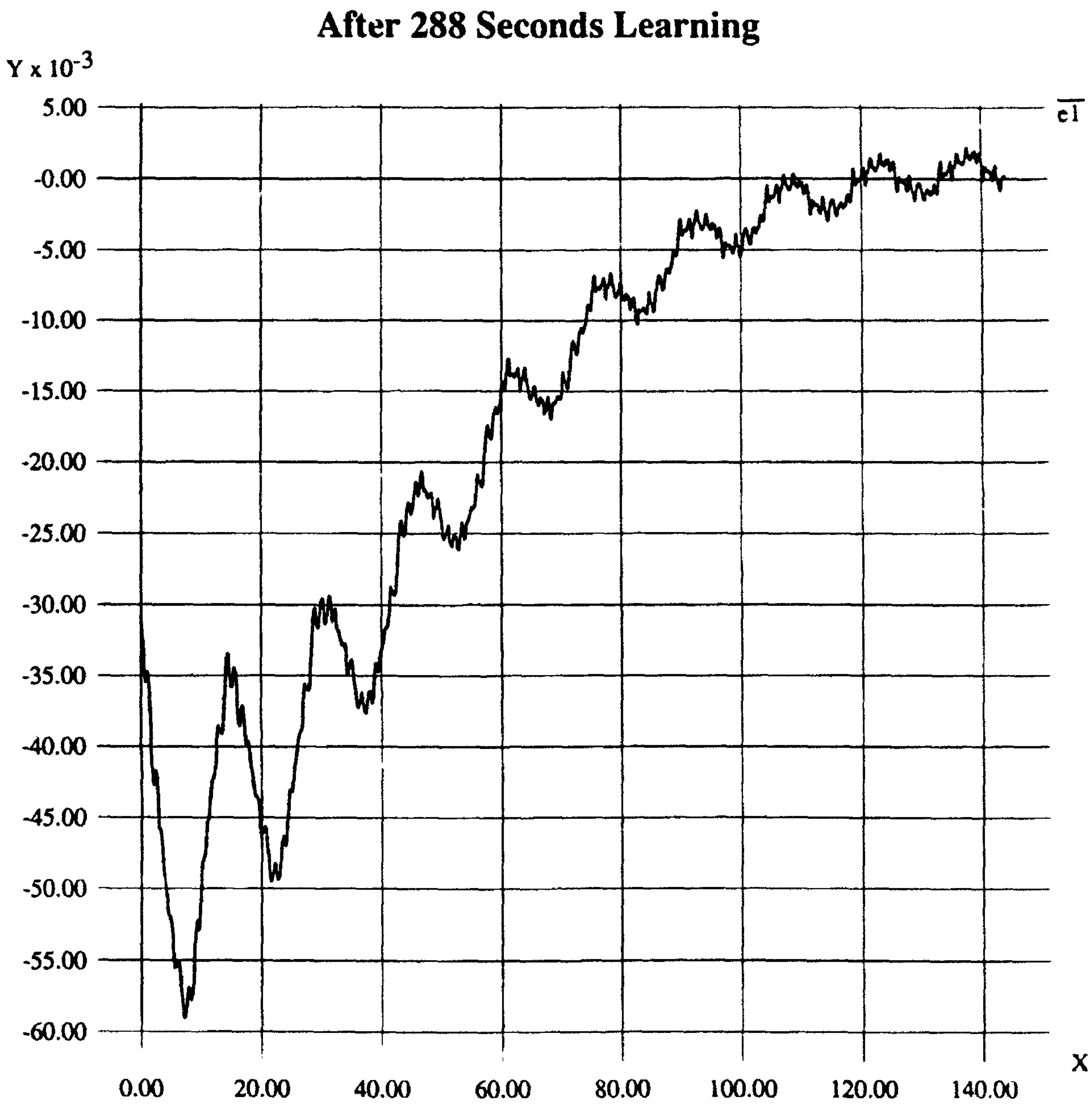

Figure 4.3-2d. Output of the D-R Neuron for the Elbow Joint (Continued). This is the second 144-second interval of learning.

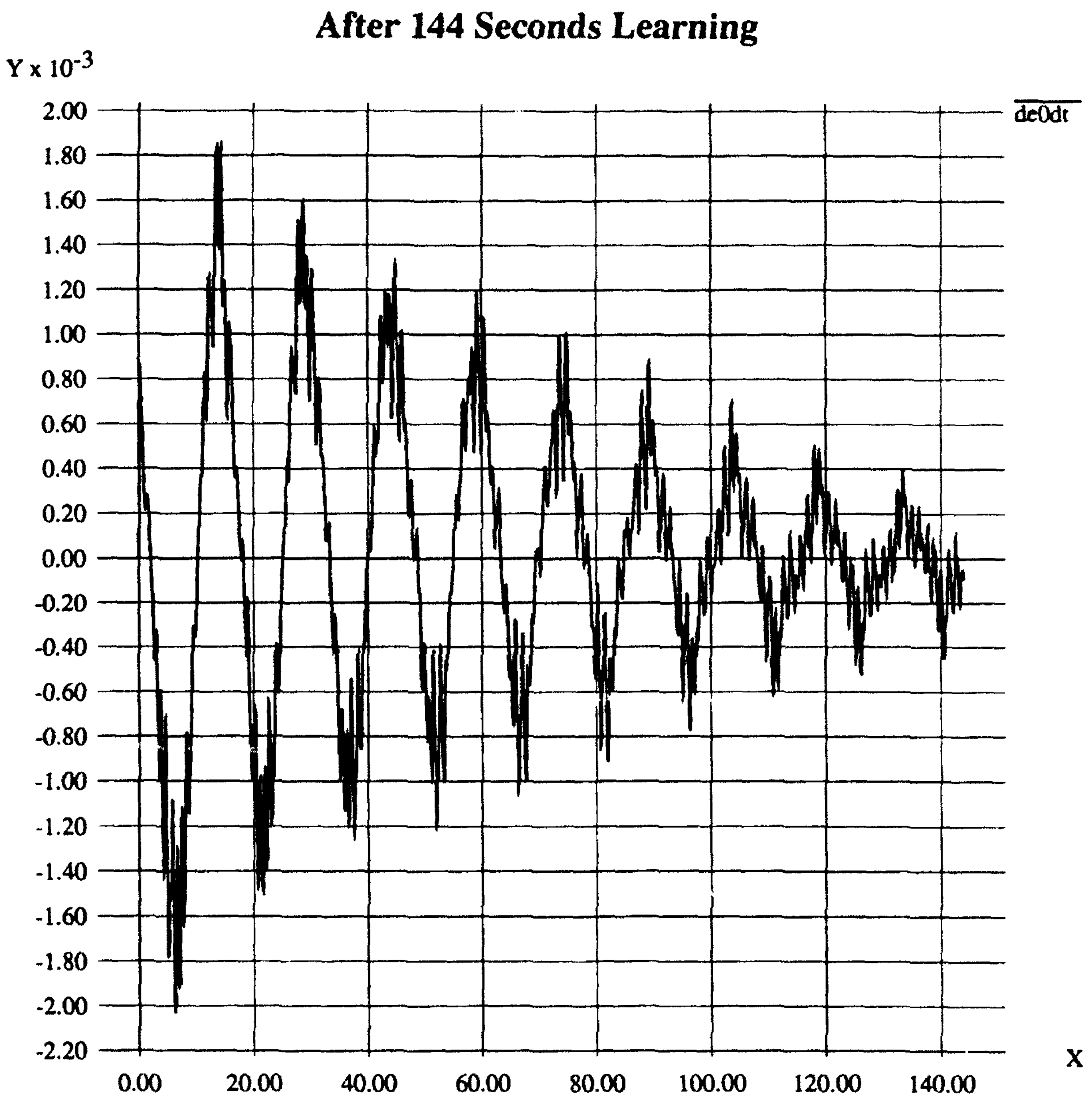

Figure 4.3-3a. Change in the Output of the D-R Neuron for the Upper Arm Joint. This is the first 144-second interval of learning.

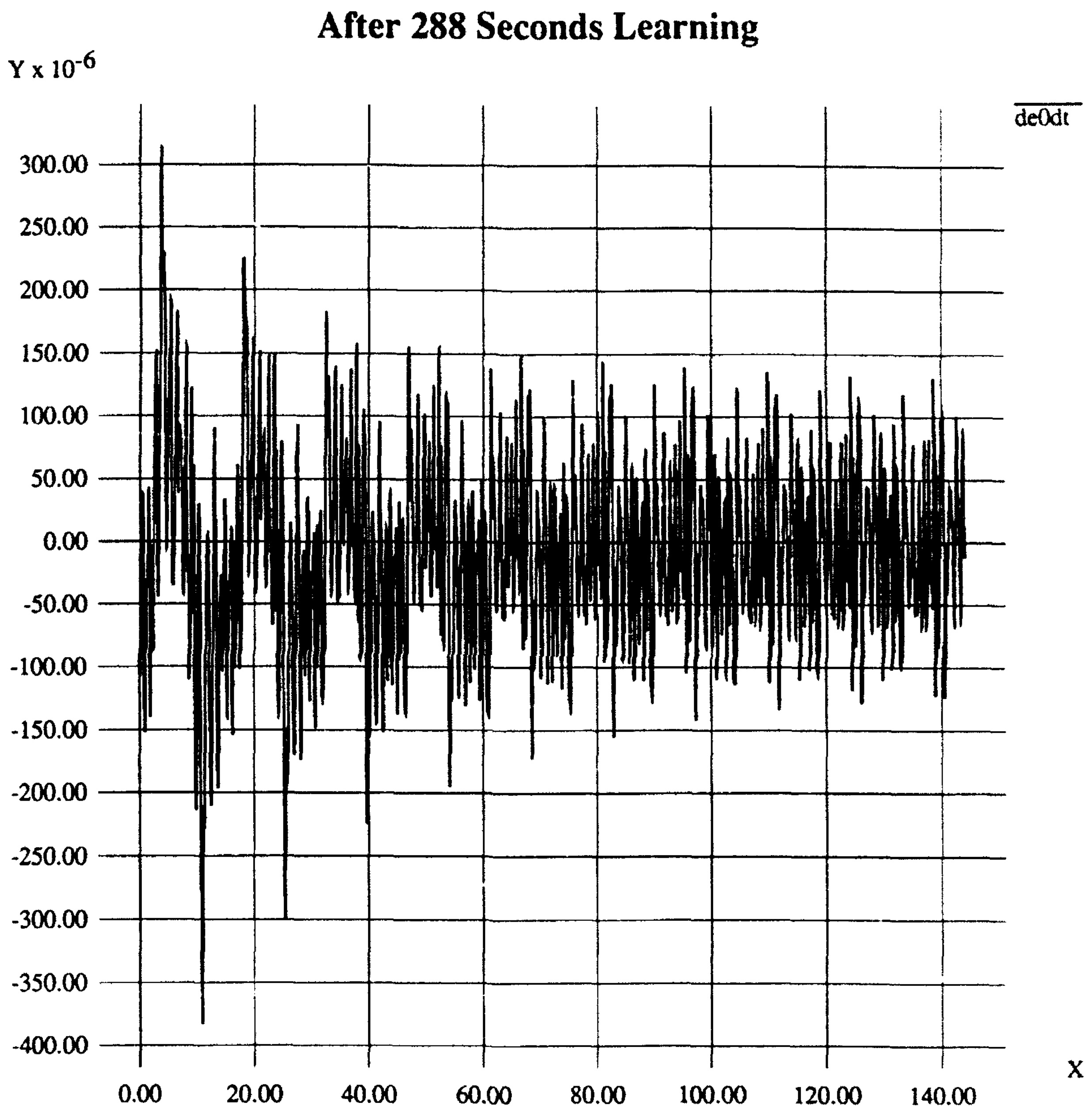

Figure 4.3-3b. Change in the Output of the D-R Neuron for the Upper Arm Joint (Continued). This is the second 144-second interval of learning.

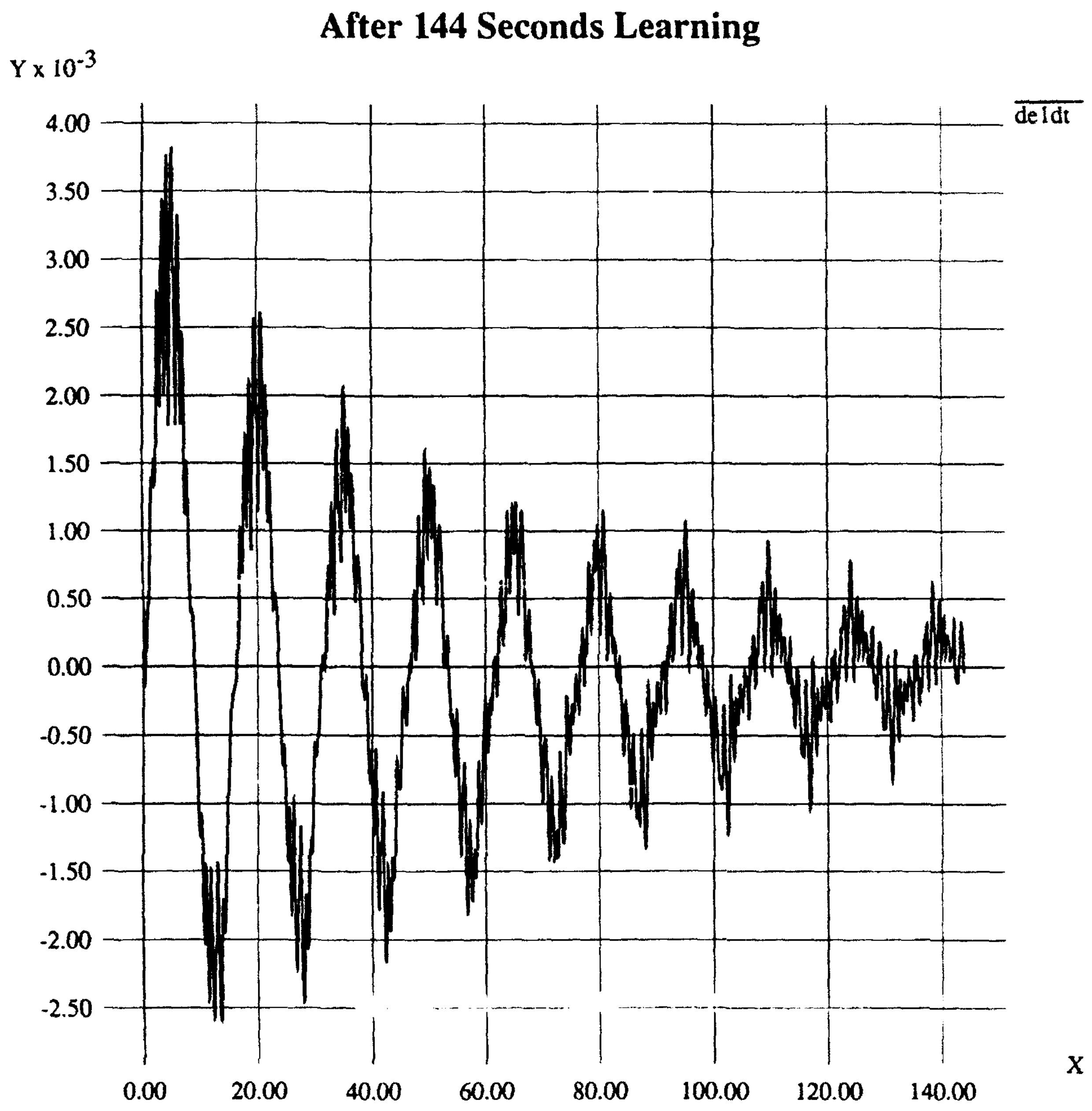

Figure 4.3-3c. Change in the Output of the D-R Neuron for the Elbow Joint. This is the first 144-second interval of learning.

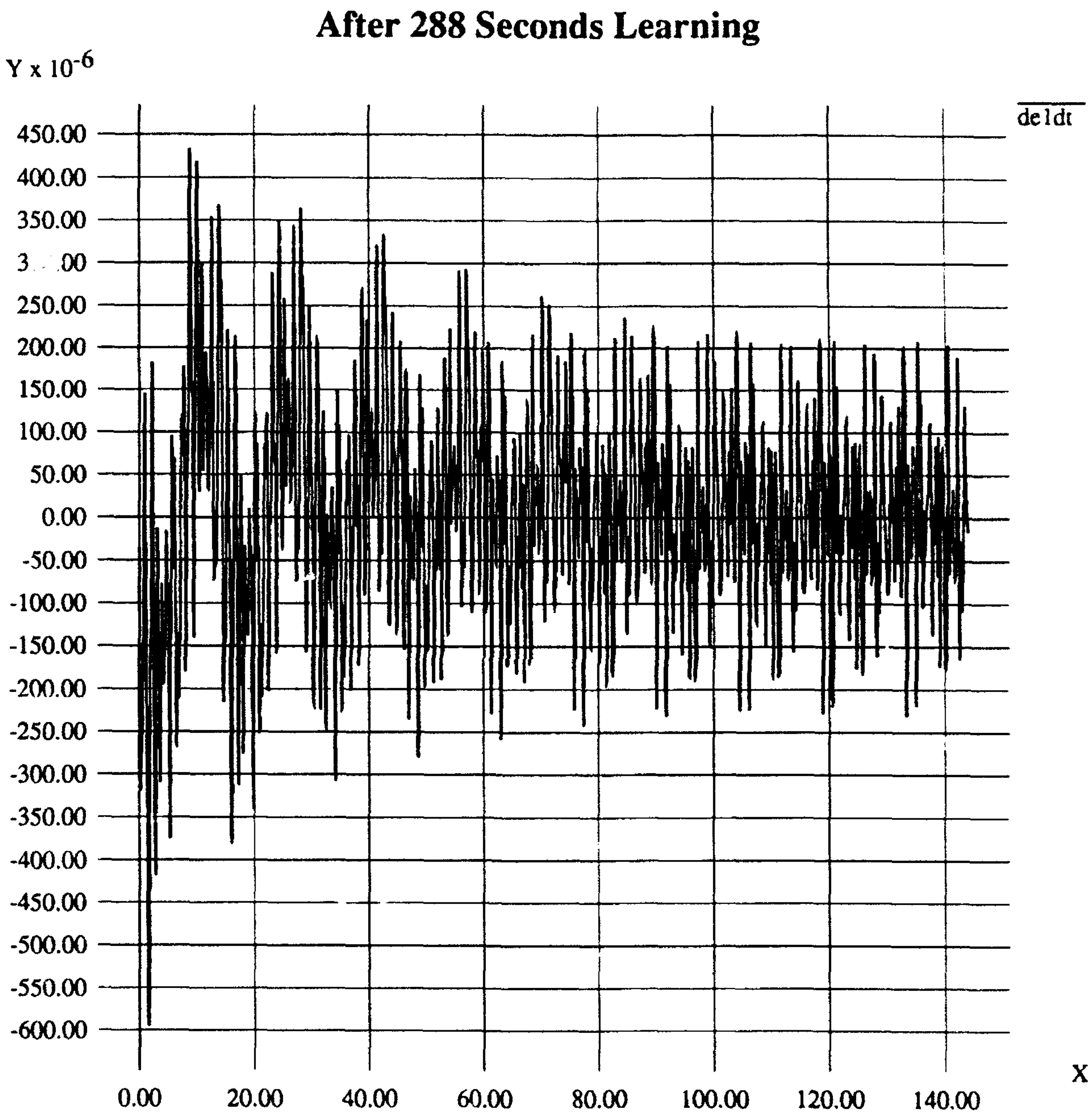

Figure 4.3-3d. Change in the Output of the D-R Neuron for the Elbow Joint (Continued). This is the second 144-second interval of learning.

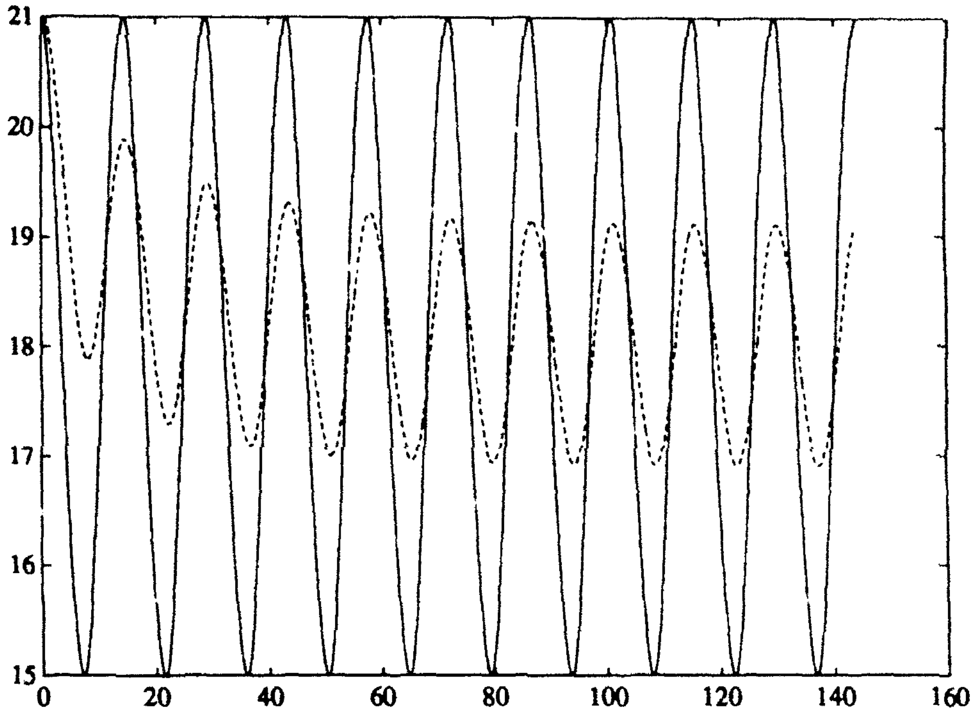

Figure 4.3-4a. Commanded and Sensed X-Axis Positions Without Learning.

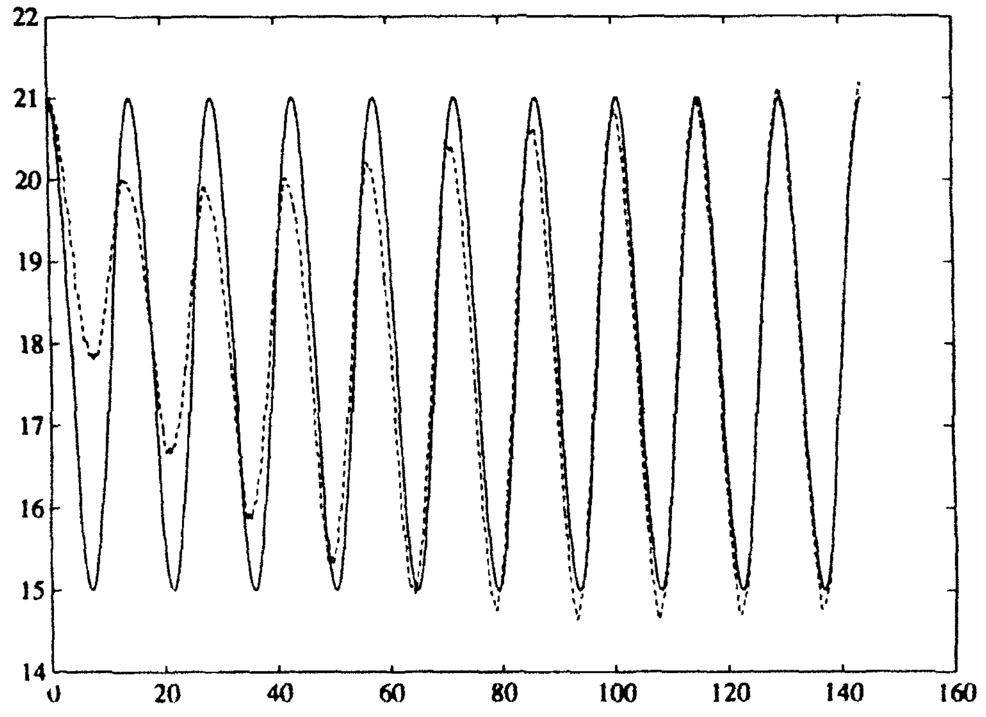

Figure 4.3-4b. Commanded and Sensed X-Axis Positions With Learning.

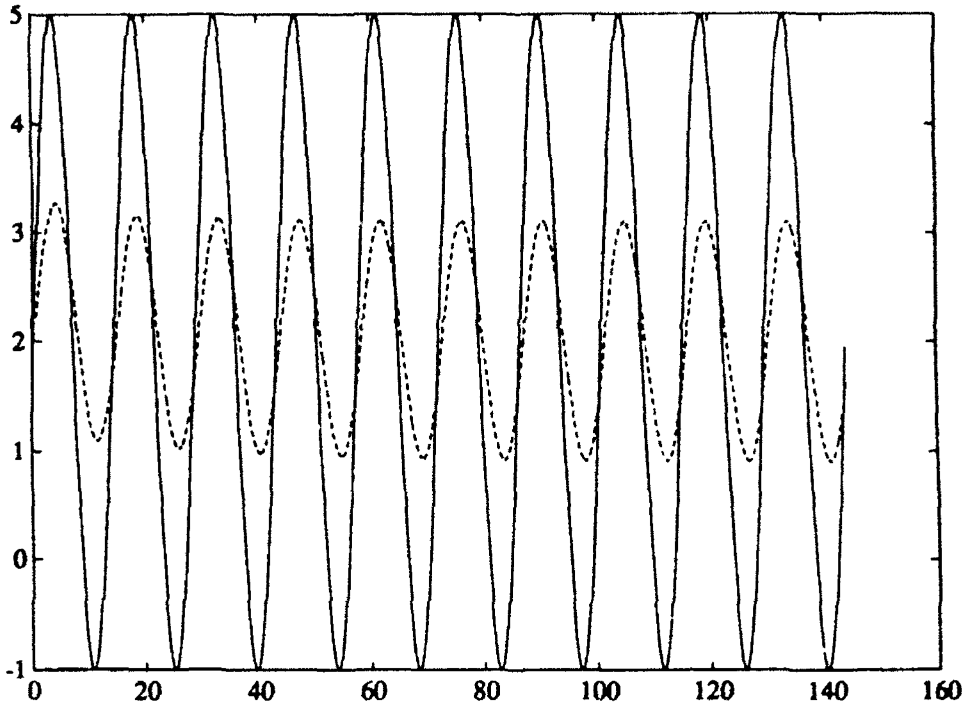

Figure 4.3-5a. Commanded and Sensed Y-Axis Positions Without Learning.

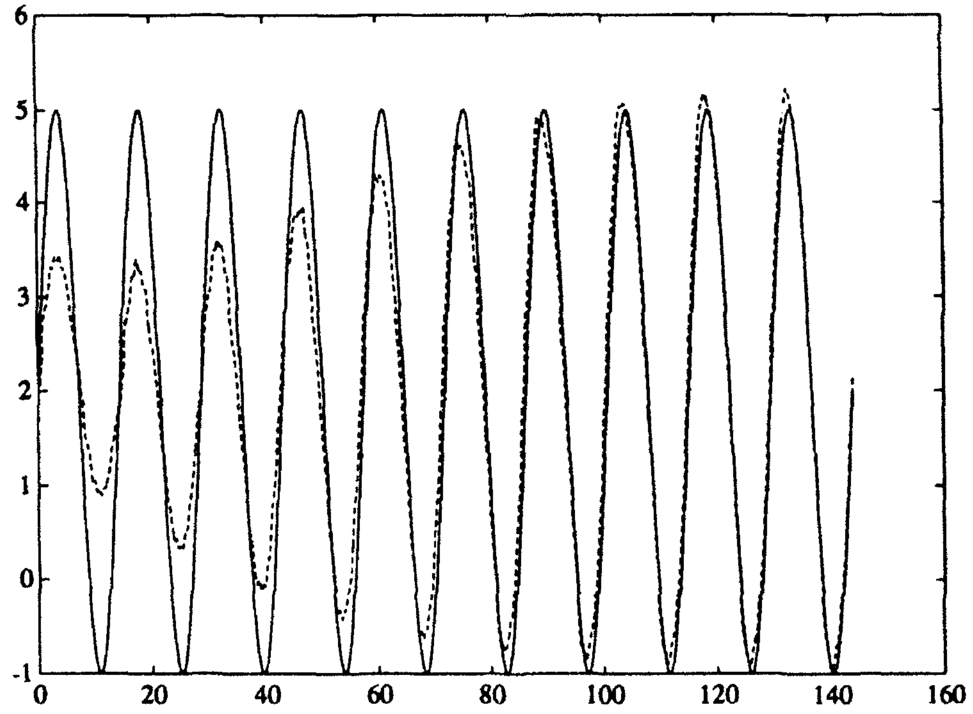

Figure 4.3-5b. Commanded and Sensed Y-Axis Positions With Learning.

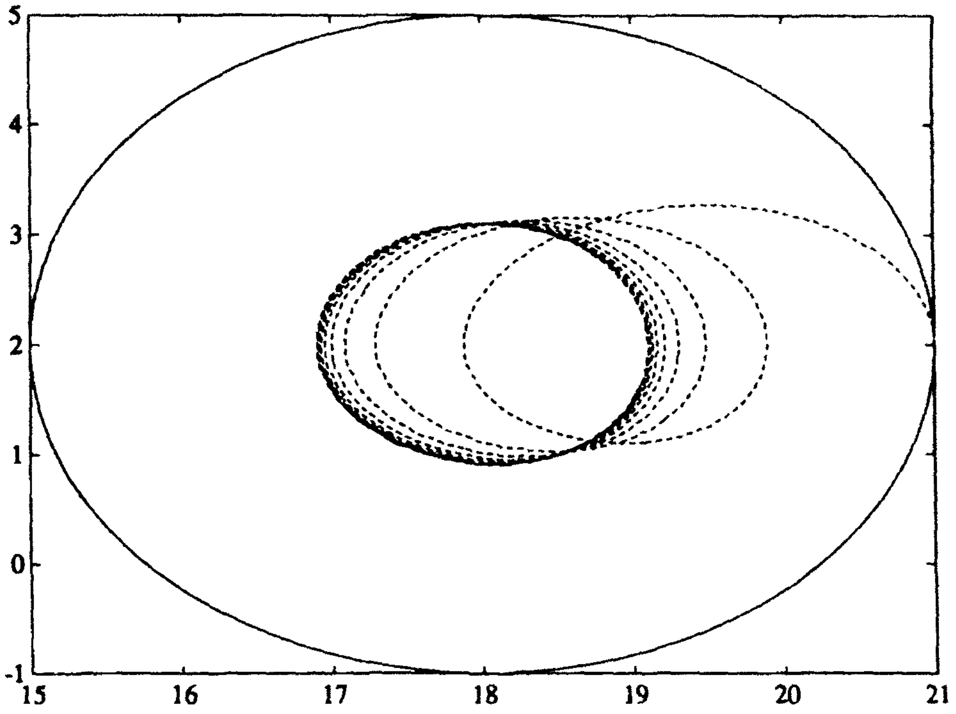

Figure 4.3-6a. Commanded and Sensed X-Y Positions Without Learning.

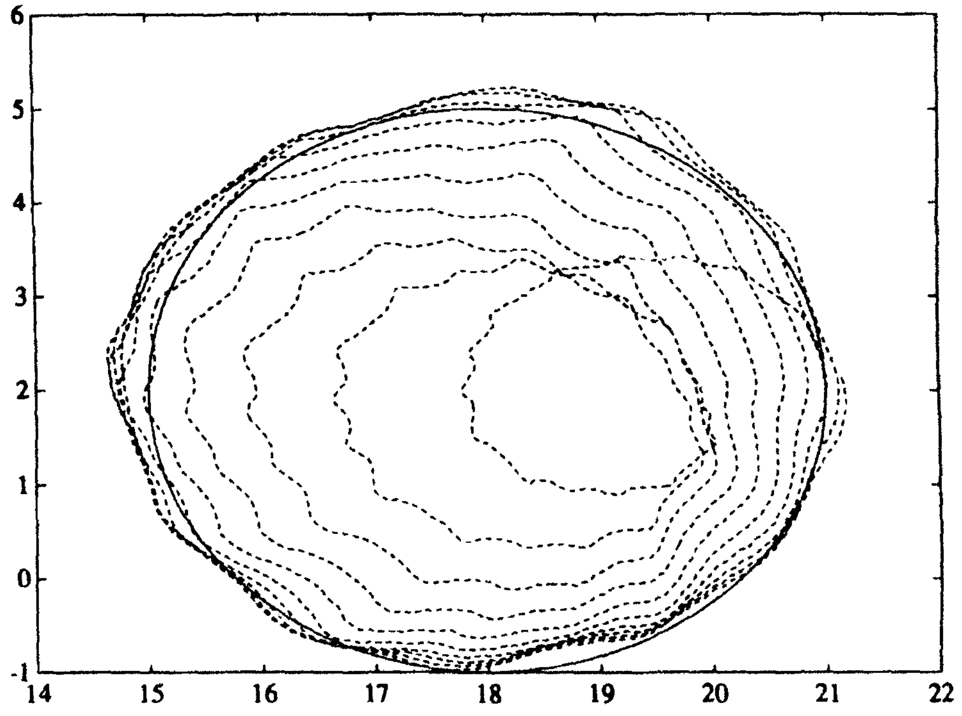

Figure 4.3-6b. Commanded and Sensed X-Y Positions With Learning.

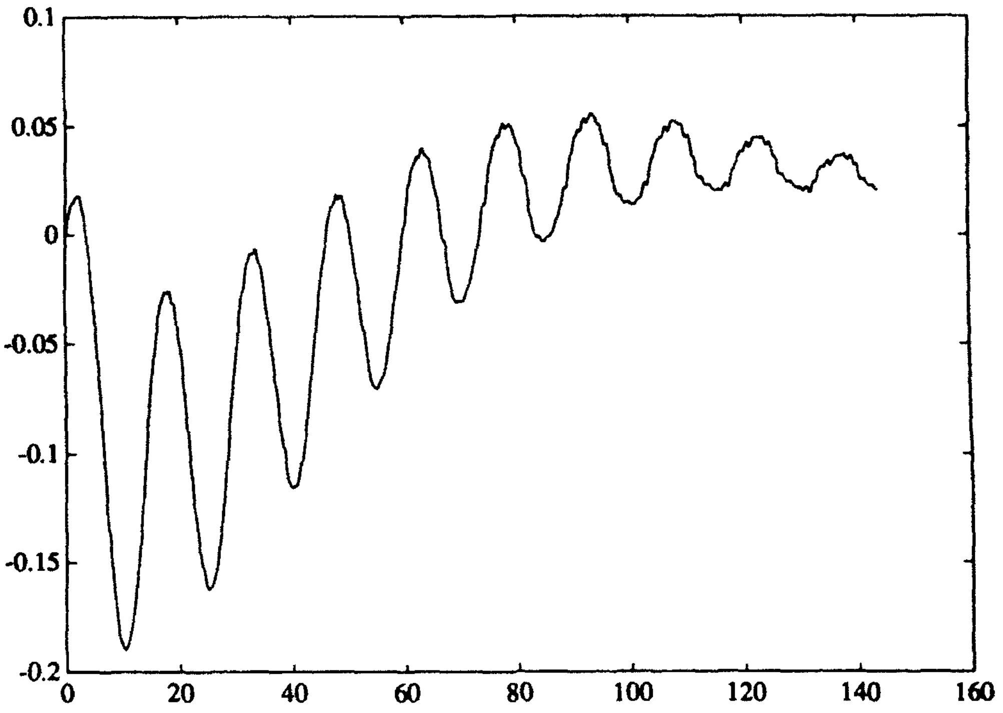

Figure 4.3-7a. Output of the D-R Neuron for the Upper Arm Joint.

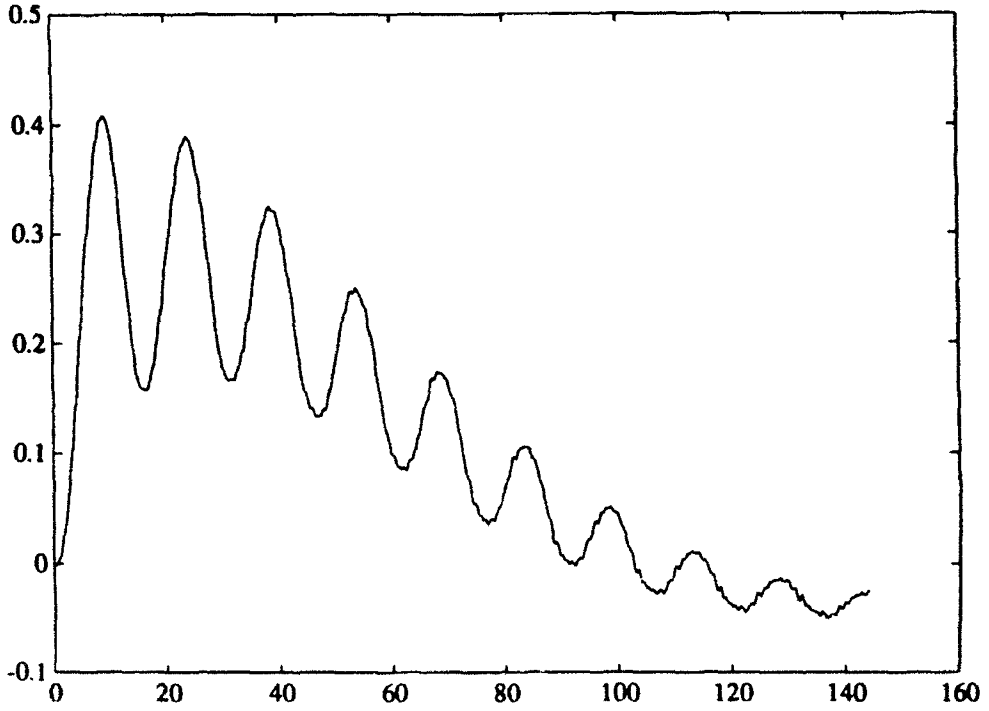

Figure 4.3-7b. Output of the D-R Neuron for the Elbow Joint.

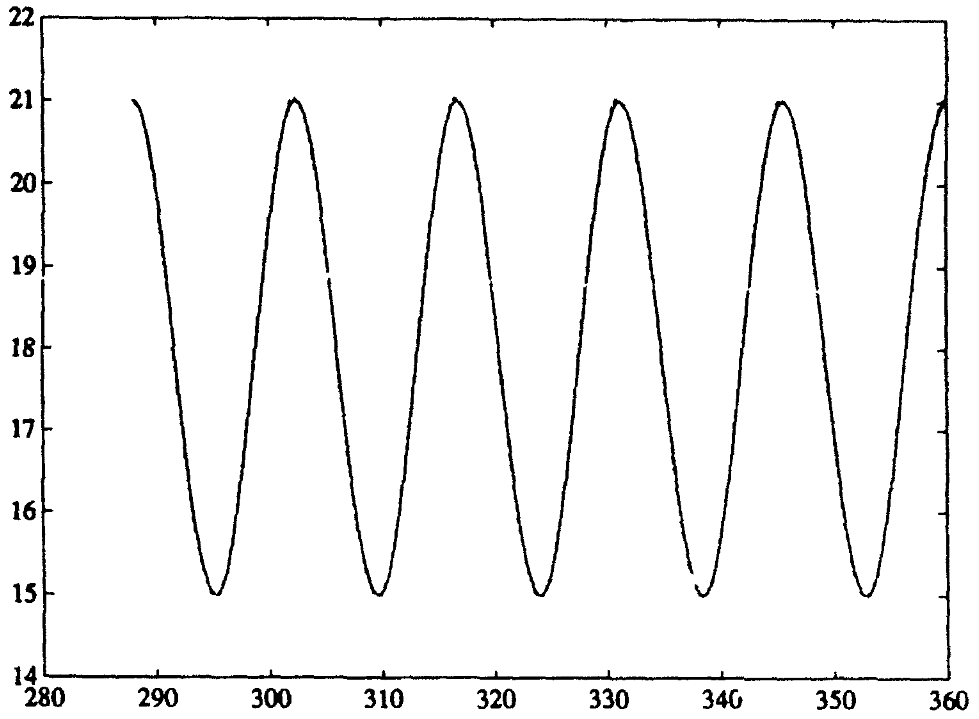

Figure 4.3-8a. Commanded and Sensed X-Axis Positions With Continued Learning.

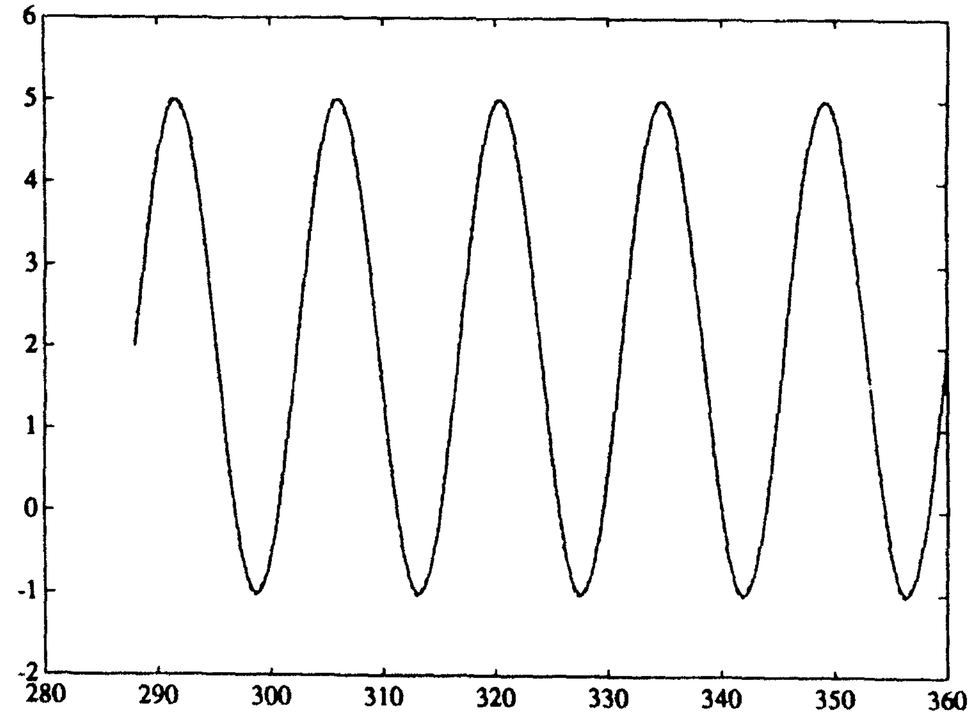

Figure 4.3-8b. Commanded and Sensed Y-Axis Positions With Continued Learning.

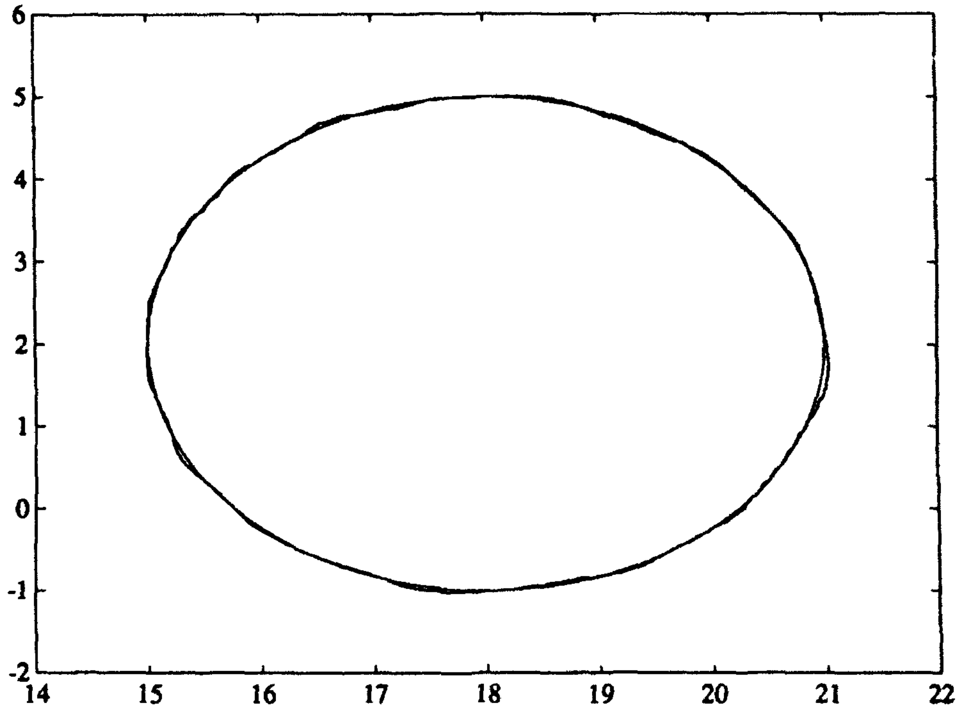

Figure 4.3-8c. Commanded and Sensed X-Y Positions With Continued Learning.

5. Conclusions and Recommendations

Our results indicate that Drive-Reinforcement (D-R) based neural networks can be successful at learning control functions in real-time with hardware. We demonstrated, for example, the ability of a D-R neuronal system to learn servo-level and trajectory-level controls for a robotic mechanism. Since our control architectures are generic, this suggests that D-R would also be effective in a broad class of control system applications outside the robotics arena. Because D-R is an unsupervised learning procedure, it has the advantages of being able to make real-time adjustments for unmodeled or time-varying dynamic and kinematic effects that model-based controllers ignore. Moreover, its 'slope matching' ability makes it very different from the standard 'function matching' neural networks commonly used. Thus, D-R may be preferred for feedback law adaptation because it is the construction of gains and not function values which are the most important and common component of the direct feedback path.

In the course of this work, we have described convergence properties of the D-R learning mechanism which guide implementation strategies, and we have specified network topologies and receptive field structures which allow for the learning of robust control laws. We recommend the continuation of both theoretical and experimental work on D-R because we believe it is at a point in its development where many benefits can be obtained in a short period of time. Particular emphasis should be placed on the development of modifications which accelerate convergence, the use of D-R in pure feedback control strategies, and additional applications of D-R to real world sensor-based control problems.

References

Albus, J. S. (1975). A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC). *J. of Dynamic Systems, Measurement, and Control*, **97**, 220-227.

Kawato, M., Uno, Y., Isobe, M., & Suzuki, R. (1988). Hierarchical neural network model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, **8**(2), 8-16.

Klopf, A. H. (1988). A neuronal model of classical conditioning. *Psychobiology*, **16**(2), 85-125.

Lane, S. H., Handelman, D. A., & Gelfand, J. J. (1992). Theory and development of higher-order CMAC neural networks. *IEEE Control Systems Magazine*, **12**(2), 23-30.

Miller, W. T., Glanz, F. H., & Kraft, L. G. (1987). Application of a general learning algorithm to the control of robotic manipulators. *The International Journal of Robotics Research*, **6**, 84-98.

Schumaker, L. L. (1981). *Spline Functions: Basic Theory*. New York: Wiley.

Strang, G. (1986). *Introduction to applied mathematics* (p. 376). Wellesley, MA: Wellesley-Cambridge Press.

Sutton, R. S., & Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In M. Gabriel and J. Moore (Eds.), *Learning and computational neuroscience: Foundations of adaptive networks* (pp. 497-537). Cambridge, MA: MIT Press.

U.S. GOVERNMENT PRINTING OFFICE 750-113