Winter 1992

# A neural network-based trajectory planner for redundant systems using direct inverse modeling

Franklin J. Rudolph
*University of New Hampshire, Durham*

# A neural network-based trajectory planner for redundant systems using direct inverse modeling

Rudolph, Franklin J., Ph.D.

University of New Hampshire, 1992

# A Neural Network Based Trajectory Planner for Redundant Systems Using Direct Inverse Modeling

BY

Franklin J. Rudolph
B.S., University of South Alabama, 1976

DISSERTATION

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy
in
Engineering

December, 1992

This dissertation has been examined and approved.

[illegible]

Dissertation Director, W.T. Miller III
Professor of Electrical Engineering

Filson H. Glanz

Filson H. Glanz, Professor of Electrical Engineering

L G Kraft

L. Gordon Kraft, III, Professor of Electrical Engineering

Michael J Carter

Michael J. Carter, Assistant Professor of Electrical Engineering

[illegible]

Lee Zia, Associate Professor of Mathematics

[illegible]

Roy M. Turner, Research Assistant Professor of Computer Science

10/5/92

October 5, 1992

## *DEDICATION*

To Abby for her patience and support, and for keeping her sense of humor throughout all this.

To Josh and Alex, just for being good kids, and somehow accepting the idea of a dad who has been in school for 22 years!

## *ACKNOWLEDGMENTS*

I would like to acknowledge my deep and abiding gratitude to Tom Miller for his faith in my abilities, Mike Carter for his encouragement, Fil Glanz and Lee Zia for inspiring me and giving me teaching role models I can look up to, to Roy Turner for his acute editorial skills and for agreeing to help out in spite of his crushingly busy schedule, and to Gordon Kraft for helping me keep this often painful process in perspective.

I would also like to thank Bruce Segee and Edgar An for their excellent groundwork in CMAC theory and fault tolerance studies and Chris Aldrich for his implementation of a protected mode 80386 assembly language CMAC model used in the Robotics Lab. Thanks also to Bart Mel and his creation, MURPHY, and to Michael Jordan from whose works I received much inspiration.

I would like to thank James Houk for patiently listening to my ravings and seeing some nugget of worth therein, and for striving for a larger context in which work like mine may someday fit.

Finally, I would especially like to express my thanks on behalf of the entire engineering community to James Albus for his insightful invention, the CMAC, whose immense importance to the fields of robotics and control theory is only just beginning to be appreciated.

This dissertation was supported, in part, by DARPA/ONR grant N00014-89-J-3100 and by an Analog Devices, Inc. Career Development Professorship.

This dissertation was produced using Ventura for Windows, version 4.1, set in 12 point New Century Schoolbook type. Graphics were generated using CorelDraw! version 2.0, QuattroPro version 2.0, and MathCAD version 3.1 for Windows. The program was developed in Borland C++ version 2.0.

# PREFACE

This thesis investigates control methods for robotics that are applied within a hierarchy. The nature of the robot hierarchy is assumed to be "intelligent" in some sense. Each level may be termed adaptive. The difference between "adaptive" and "intelligent" is intended by this author as a matter of legitimate difference in degree rather than mere semantics.

The term "adaptive" may be seen as connoting such attributes as "compliant" or "adjustable". The conglomerate result of many separate adaptive modules or agents acting in concert as has been postulated by Minsky ['86], Brooks ['88-91a], Hofstadter ['79], and others is that some activity resembling intellect can emerge from the synergistic combinations of the activities of all these modular parts. The agents differ in these various authors' methods, but the sense each author conveys is that the combined result is somehow greater than the sum of the parts.

Minsky and Hofstadter present a descriptive approach explicative of colonies of organisms they have observed, while Brooks takes a more "ontogenic", or bottom-up approach. Brooks claims that, in essence, just by doing what we engineers do best, i.e., by being careful to do the right thing at every step, and by starting at the very bottom we can not help but eventually invent a robust, capable, more or less autonomous system. The resultant system may not be very "smart" except in some primitive but perhaps important way, e.g. in survival skills.

It is the addition of the descriptor "autonomous" that starts to describe what is meant herein by the difference between adaptive and intelligent.

In all 3 systems mentioned, a global world model is not important to the emergence of intelligence. Brooks takes the extreme view, Brooks [91a], in that he *demands* no representation of the external world be implemented. Minsky and Hofstadter on the other hand just do not require it. Brooks holds that planners at every level up to and including the cybernetic "pilot" of the system can be removed and replaced with some highly complex collection of layered agents. The agents in Brooks's hierarchy are cunningly interconnected by subsumptive and inhibitory communications relays. One could well argue that a Brooksian system does in fact have a world model embedded in it, and the layout, enumeration and interconnection of these relays is an implementation of it. This view begs the question of just where the data and the algorithm lie in such a system. An extreme Brooksian scholar might conversely argue that an AI implementation of an intelligent system is equivalent to some Brooks-style subsumption implementation in that it is implemented with millions of low level switches and logic gates that implement the opcodes comprising the computation and inferences of the system, and that there is a ridiculous superfluity of them due to the inefficiency of a lot of semantic fluff implemented high up in the system at the LISP or PROLOG level.

These two are extreme views. Bellingham and Consi [90, 91], Schudy and Duarte ['90], Bellingham and Beaton ['89] argue for a middle approach.

The objection to Brooks's view that applies here is more a convenience than a power issue. Consider, for instance, that a Turing machine has the same computational *power* as a modern computer with a C++ compiler. It is nonetheless ludicrous to argue that the former is a practical replacement for the latter. It seems that though a purely subsumptive control system might be implementable, given enough insight into the microstructure of the behavioral fabric of a particular, highly complex system woven by the Brooksian behaviorist, the *convenience* of a system built by subsumptive revisionists like Bellingham *et al* (or *this* author) is critical, thereby allowing programmability in ways that the purely subsumptive system disallows

Clearly, subsumptive style systems have some enticing and useful attributes that make the systems robust and compliant. Each layer is independent and sufficient for the generation of a behavior (though not necessarily the *right* one), so a failure at a high level does not stop the creature in its tracks; it just might make it behave in a less adequate fashion. The layers are more relaxed than say a traditional computer network, or a parallel sort algorithm. It would be entirely sufficient for perhaps 10% of the packets sent by one layer to reach their destinations at lower layers. Thus, large amounts of time and effort on the part of the system are not wasted on handshaking. The effect of this is that higher layers do not command lower ones, they simply attempt to influence them.

Top down is not the only direction of data flow. Albus ['81] and Houk ['88] both argue for ascending data flows in their two quite different cerebellar models. A combined strategy consisting of ascending and descending data flows through a complex multi-level system is much more like the leaky but adequate neural channels for control found in nature than in most engineering solutions. Such a technique is highly robust. Other techniques, like carefully tailored analytical ones can suffer from brittleness, a lack of adaptability, and even stability problems in the face of long and variable time delays.

## TABLE OF CONTENTS

## *LIST OF FIGURES*

## *ABSTRACT*

## *A NEURAL NETWORK BASED TRAJECTORY PLANNER FOR REDUNDANT SYSTEMS USING DIRECT INVERSE MODELING*

*by*

***Franklin J. Rudolph***
***University of New Hampshire, December, 1992***

Redundant (i.e., under-determined) systems can not be trained effectively using direct inverse modeling with supervised learning, for reasons well outlined by Michael Jordan at MIT. There is a "loop-hole", however, in Jordan's preconditions, which seems to allow just such an architecture. A robot path planner implementing a cerebellar inspired "habituation" paradigm with such an architecture will be introduced. The system, called ARTFORMS, for "Adaptive Redundant Trajectory Formation System" uses on-line training of multiple CMACS. CMACs are locally generalizing networks, and have an *a priori* deterministic geometric input space mapping. These properties together with on-line learning and rapid convergence satisfy the loop-hole conditions. Issues of stability/plasticity, presentation order and generalization, computational complexity, and subsumptive fusion of multiple networks are discussed.

Two implementations are described. The first is shown not to be "goal directed" enough for ultimate success. The second, which is highly successful, is made more goal directed by the addition of secondary training, which reduces the dimensionality of the problem by using a set of constraint equations. Running open loop with respect to posture (the system metric which reduces dimensionality) is seen to be the root cause of the first system's failure, not the use of the direct inverse method. In fact, several nice properties of direct inverse modeling contribute to the system's convergence speed, robustness and compliance.

The central problem used to demonstrate this method is the control of trajectory formation for a planar kinematic chain with a variable number of joints.

Finally, this method is extended to implement adaptive obstacle avoidance.

# *INTRODUCTION*

The broad discussion of the preface infers that actions apparently intelligent or purposeful can be synthesized by composition of a collection of rather primitive activities. The complex activity this dissertation will discuss is the activity of a robotic manipulator arm. This is a classic control problem with many applications in manufacturing, hazardous waste management, assembly and manipulation of structures in space, etc.

In order to meaningfully compose primitive behaviors into complex ones, we must first decompose complex ones conceptually to know which tasks to assign to individual modules in an overall system. The simplest such decomposition is to grossly decompose the complete process into three levels:

- TASK level or intentional level: This level decides what work space objects to manipulate, and in what order, and assigns spatial coordinates to the target objects.
- Primitive or elemental move level (PRIM/EMOVE): This level figures out how to form trajectories. It decides how to move the end effector of the arm from one ultimate target to another, and perhaps elects intermediate targets between the TASK appointed ones. From here, joint angle trajectories are output to the lowest level.
- SERVO level: This level turns desired joint space commands into moves of the arm. It is much like a traditional tracking controller.

The method developed here the activity of the middle level. Its principal task is to find a meaningful mapping from the target space, i.e., locations of things to be manipulated, to the joint space, the space wherein the direct means of controlling the articulator are embedded. Many such mapping methods have been proposed. (See section 1.2). Some are analytical methods which are computationally expensive and require exhaustive knowledge of the analytical form of the model. Others are neural network based, and though they don't require detailed analytical model explication, are computationally expensive because of the nature of the neural network models used. (See section 2.6).

Thus the middle level task decomposer niche is ripe for an implementation which purports to solve its problem in a fashion that is

- fast enough to be capable of real-time operation, and
- not dependent on knowledge of the analytical form of the plant model.

The high level (TASK) decomposer is not treated here because it may not be implementable by means other than symbolic methods requiring the deep recursive (back-track) search of logic programming, formal languages and artificial intelligence.

Barto ['89] subdivided modern adaptive methods into two classes:

- vector space methods and
- articulated methods.

Connectionist methods are of the former and symbolic AI methods are of the latter type.

Some limited back-track search capabilities are embodied in Sutton's temporal difference method, and in a method proposed here, called temporal lookahead method. (See section 6.4.2.2). Both these are arguably connectionist methods, but they are certainly vector space methods. At this time these methods don't appear to be general enough to implement the deep search necessary to tackle the TASK level, so the TASK level remains outside the scope of this work.

The SERVO level is also not treated here because that level has been fully treated by Miller *et al* ['86-'92].

So the purpose of my system is to perform just the middle level task decomposition and to produce a stream of data to send to a SERVO level system based on input it receives from a higher level TASK module both of which are external to this development.

Here, we shall discuss that the system's task be performed after a fashion of simply trying things in a constrained trial and error method wherein the results of nearly all trials become training instances whereby the system incrementally learns how to accurately predict what joint moves are necessary to accomplish desired work space moves suggested by the TASK level.

What makes this mapping difficult to achieve is that the class of articulators we shall discuss are redundant ones, for which no analytical means of computing the required work space to joint space mapping in closed form exists. (See section 1.1.2). The analytical method is then doomed to some computationally burdensome iterative process unless a fast "reflexive" method like the one about to be proposed can suffice. (See page 10 and section 3.4 for discussion of the term, reflexive).

We shall discuss a neural network based adaptive method rather than the well known linear adaptive methods (which would also provide a fast "reflexive" implementation), because the kinematic plant model is too non-linear for the latter to suffice.

There is a widely accepted view concerning redundant inverse models, which holds that training should not come from direct experience because there are potentially conflicting experiences in which differing inverse model inputs can generate the same inverse model outputs. These differing inputs may be widely disparate and non-linearly related, so their averages may actually not even *be* solutions.

On the other hand, direct training of inverse models (or controllers) for non-redundant systems has proven to be fast and effective, making it desirable to use similar techniques for redundant systems.

This dissertation will show that direct training during on-line learning, with heuristic guidance, can give fast effective adaptation for redundant systems, thereby avoiding the pitfalls just described.

In addition, we shall see that different heuristics (i.e. joint motion constraints) can lead to different solutions in a robot's joint space which satisfy identical requirements in the hand space. Taking advantage of this property, a method will be developed in which spatially varying heuristics can be stored that can be used to provide different motion characteristics (in the joint space) for different regions of the operating space. Finally, this method will be applied to the problem of work space obstacle avoidance.

Chapter 1 discusses other authors' approaches to similar problems.

Chapter 2 discusses general issues related to these other approaches and how these issues guided me to the current development.

Chapter 3 describes an initial approach toward the end of direct inverse learning for redundant systems. This method didn't work out well, but it helped refine basic concepts which were later exploited successfully in the modified technique described in chapter 4.

Chapter 5 describes the basic obstacle avoidance problem and how it can be approached by modifying training heuristics.

Chapter 6 discusses near term improvements and broad conclusions.

Chapter 7 is a synopsis of direct conclusions from the main body of the dissertation and a discussion future work.

Appendix A is a more speculative and long term discussion of future directions for this and related adaptive systems.

Several other appendices are included as support for various arguments in the text.

# *Chapter I*

# *Background*

## 1.1 What Makes a Good Trajectory Planner?

### 1.1.1 The Middle Level Of Control

Path planning for robotics is a particularly thorny problem, in that it is difficult to define. Wavering *et al* ['88] represent path planning as 3 levels: world model/task decomposition (TASK) elemental move level (E-MOVE) and primitive level (PRIM). If one constructs a hierarchical model composed of these three modules, one will have all of the brain functions emulated above approximately the brain stem/spinal cord reflex level of a complex organism. The most primitive reflex or servo level (SERVO) would then complete the system.

It is doubtful at the current level of artificial neural network development that the TASK level, which might also be called the "intentional" level, can be implemented using other than "traditional" (symbolic) AI techniques. Houk ['90] and Albus ['81] have each proposed cerebellar models of relevance to the lower and middle hierarchical levels. Houk talks about the cerebellum as an array of adjustable central pattern generators. This model gives insight into the nature of E-MOVE and PRIM level path planning activities. Albus has proposed and implemented a simple CMAC (Cerebellar Model Arithmetic Computer) neural network model that has been developed and refined here at UNH. The CMAC model has proven itself to be extremely computationally efficient and exhibits adaptive properties. CMAC has proven to be an excellent adaptive element for implementing SERVO level control functions.

In this dissertation, we shall discuss path planning as defined at the PRIM level. This is exclusive of "intentional level" processing and the SERVO level processing as described by Wavering ['88], but would include some of the processing proposed therein at the E-MOVE level. Notably, the inverse kinematics and redundancy resolution will be handled in this proposed adaptive model.

### *1.1.2* ***Varieties of Redundancy***

A robot manipulator arm is a physical realization of a *kinematic transformation:*

$$\mathcal{K}: \mathcal{R}^n \rightarrow \mathcal{R}^m.$$

Under the appropriate constraints there may exist an *inverse kinematic transformation:*

$$\mathcal{K}^{-1}: \mathcal{R}^m \rightarrow \mathcal{R}^n.$$

These transforms describe the response of the manipulator to joint postures and joint movements.

The "joint space", $\mathcal{R}^n$, uniquely defines the posture of the arm, and coincidentally uniquely defines the hand position[1]. There are $n$ joints in the kinematic linkage, and the hand space is $m$ dimensional. One could then express the forward kinematics as $X_h = \mathcal{K}(\Theta)$.

The inverse kinematics, expressed as $\Theta = \mathcal{K}^{-1}(X_h)$, is a bit more problematic. The manipulator's hand or actuator coordinates, $X_h$, reside in the *hand space* or *work space*. The hand space, $\mathcal{R}^m$, uniquely defines the hand position, but not necessarily the arm posture. The hand space may be 2 or 3 dimensional, de-

1 It is possible to define more complex hand spaces, for instance in the cases where the hand has additional degrees of freedom, like orientation or the degree to which a terminal manipulator is open or closed. In this dissertation, only 2 or 3 dimensional positioning of a "point hand" will be considered.

pending on whether the arm linkage is planar.[2] The issue of *linkage redundancy* addresses whether or not this inverse kinematic transform exists.

For any configuration and any values of $m,n \neq 0$, given $\Theta$, $X_h$ is uniquely defined. If $m = n$ the arm is non-redundant, and the inverse kinematic transform exists. If $m \neq n$, however, things get more complicated. In general, if $m < n$ the arm is said to be redundant. In other words, for a simple[3] linkage, whenever the number of joints exceeds the degrees of freedom of the hand, no unique inverse kinematic solution exists. Any arm is called *redundant* if no *unique* inverse kinematic solution exists.

There are actually 2 kinds of redundancies involved in manipulators: path redundancy and postural redundancy. For a path redundant manipulator, multiple paths exist between any two hand (endpoint) positions. A posturally redundant mechanism can maintain the same endpoint position for many different postures, or settings of the joints. A two link arm similar to the 3 link arm in figure 1.1 is only path redundant. A two link arm, however, that has 2 prismatic (sliding) joints as in figure 1.2, is posturally redundant. It seems obvious that the sliding joint mechanism may be less resistant to analysis than one with rota-

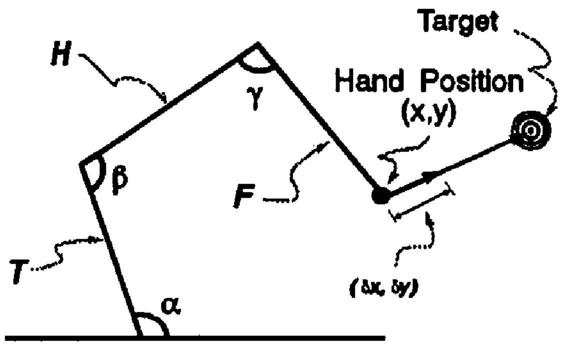


Figure 1.1: A 2D Planar Redundant Articulator

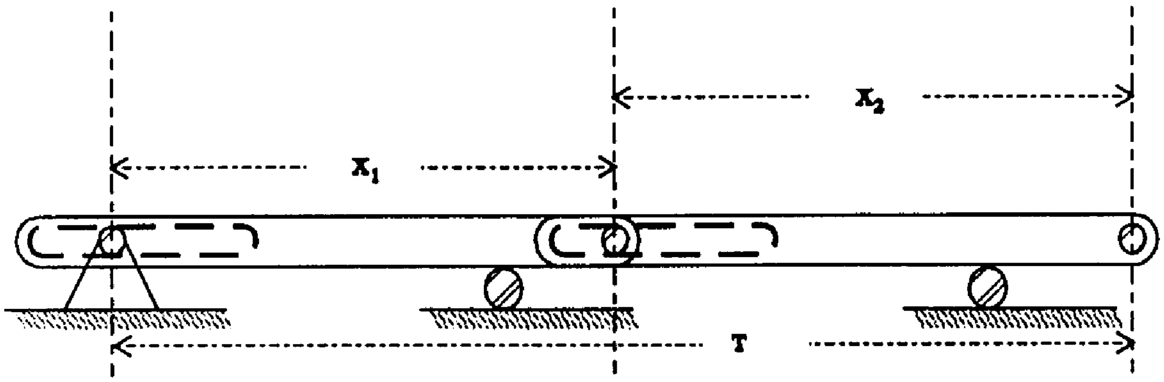
2 A planar linkage is one in which all the links and joints can be contained in a plane for all possible settings of the joint angles.

3 The term "simple" refers to a linkage in which the joints are all independent. No two or more joints can be coupled.

tional joints, and yet the two link arm with prismatic joints is posturally redundant as is the 3 link rotationally jointed mechanism. Unfortunately, however, the prismatic system is a linear system, and thus is not very interesting.

Since nearly all practical manipulators are path redundant, for the rest of this dissertation, the term *redundant*, unless specified otherwise, will denote *postural* redundancy.

A redundant arm system both poses and solves problems. The problem it solves is that with postural redundancy, an infinite number of postures of the



**The target position is a linear combination of the input coordinates.**

Figure 1.2: A Prismatic Linkage

joints is possible given a particular hand position, which gives the linkage power and flexibility in finding paths that avoid obstacles. The problem it poses is that it requires, given a hand position, that a posture be found via an iterative search or constraint satisfaction procedure, because the inverse kinematics cannot be solved in the straightforward fashion of computing a matrix inverse as is the case with a non-redundant arm.

It is possible to consider differential forms of these transforms: $\delta X_h = \mathcal{K}(\delta\Theta)$, and $\delta\Theta = \mathcal{K}^{-1}(\delta X_h)$. The "path finding" problem still applies for the differential case which is used to describe gross movements as sequences of small movements rather than as pairs of trajectory endpoint postures and the torques it takes to move from one to the other.

A trajectory specified in hand space as a series of hand positions, $\left\{X_{h_i}\right\}$, is a reasonable form of problem specification. The corresponding series in joint space $\left\{\Theta_i\right\}$ must be derived using whatever method is available. The objective in the remainder of this dissertation is to determine a method for a solution $\left\{X_{h_i}\right\} \rightarrow \left\{\Theta_i\right\}$ that will have knowledge about obstacles "reflexively" embedded in a hyperspatial representation of the transform, rather than being declarative in nature and thus requiring that we execute a search whenever a solution is required. The terms *declarative* and *reflexive* in this context are used in the same sense as in the work of Handelman, Gelfand and Lane (Handelman ['89]). This is an important concept discussed further in section 3.5.

The actual solution will be to find the incremental series $\left\{\delta\Theta_i\right\}$ that sweeps out a series of postures, to generate an incremental series $\left\{\delta X_{h_i}\right\}$, given an initial condition, $X_{h_0}$, and target position $X_{h_f}$. To develop the incremental series, we must develop inverse Jacobian matrix solutions. A neural network will be used to learn local inverse Jacobian transformations, and the property of local generalization within the network is essential for reasons that will be discussed at length.

## 1.2 Previous Work

### 1.2.1 Degrees of Optimality

The term, optimal, is a relative (and often abused) term. In the following discussion, no attempt will be made to describe precisely what quantity each method cited from the literature is purported to be optimal (or near optimal) with respect to. In some cases, like Kawato ['89, '89a], it would be simple to do so, because in his case, optimality is with respect to torque exertion. In other cases it is often not so easily stated. For instance Canney ['90], Korein ['85] and Lozano-Perez ['87] treated such broad classes of problems that the methods could be said to be optimal with respect to many different measures, depending on the implementation. In general then, let us consider that each method is optimal with respect to search effort within a model representational space.

A nearly optimal solution for a redundant arm using a stochastic method combined with heuristic search was investigated by Mel ['89, '90], in a system called MURPHY. Others have solved the optimal problem in a minimum norm deterministic sense, e.g. Klein ['83], or a variety of constrained search methods (e.g. Canney ['87] and Korein ['85]) all of which are both highly computationally intensive and difficult to set up for any particular problem. Canney's treatment of the *generalized mover's problem* attacked a much wider class of trajectory planning problems than will be discussed here.

Lozano-Perez ['87] used an $A^*$ search to find optimal trajectories after a geometric enumeration of all possible arm/obstacle configurations, but the measure of complexity was worse than exponential in the number of degrees of freedom of the arm[4], and the constraints placed on the arm and work space were perhaps too restrictive for general applicability.

Jordan ['88] used a constraint satisfaction method coupled with a stochastic "pre-search". For a more complete discussion, see sections 1.3.4 and 2.2.1.

Kawato ['89, '90] proposed a method that was optimal (with respect to exerted torque) within the scope of the simple trajectory model it implemented. The model was, however, so crude an approximation of both the time and space required for a trajectory formation that it was suboptimal with respect to most real world problems it solved. In this method time was converted to space by building (in simulation) a finite impulse response approximation of an infinite impulse response[5] closed loop system. A simulated feed-forward multi-layer perceptron (MLP) implemented each step of a trajectory (another space costly design)[6] and recurrent feed-forward and feedback loops provided constraints between time and space that can directly generated optimal torques. In his method, waypoints were explicitly clamped at any *m* of the *n* nodes in the network, and the *n-m* free nodes then developed waypoints that conformed to a minimum torque smoothness constraint. No means of acquiring *specific* desired waypoints was suggested, so his method (as well as Jordan's) is a different kind of trajectory planner than the one proposed here.

One can conclude early on in a study of the literature that a high computational cost and loss of generality are the price of an optimal method in this area!

4 His complexity measure was $O(r^{k-1}(mn)^2)$, where $r$ is the resolution of the joint encoder, $k$ = the number of degrees of freedom of the arm and $m$ and $n$ are measures of the complexity of the arm and the work space.

5 A finite impulse response (FIR) system approximates an infinite impulse response (IIR) system. An IIR is like an analog feedback controller in that every input continues to affect the output response until its effect becomes undetectable. An FIR only considers the effect of an input over a fixed, finite number of time steps.

6 This is similar to the back propagation through time method advocated by Nguyen ['89] and Williams ['89].

*1.2.1.1* ***Optimality Is Not Central To Success.*** Other authors have developed methods of obstacle avoidance for path planning applications that are suboptimal. Reinforcement learning has been exhaustively studied by Barto ['83] and Sutton ['90]. These reinforcement learning methods can be classified as suboptimal because they culminate in an *approximation* of dynamic programming, which is, when implemented exactly, an optimal method. Part of the first generation solution developed here involves a crude form of reinforcement learning that is simpler than the method of Barto and Sutton's work. In this method, a specific cost function is not modeled; rather an incremental response to a non-specific punishment signal is implemented.

Bullock ['88] developed a biologically inspired approach that synthesized some of Grossberg's neural network modeling ideas with neuromuscular junction models. In his work, it was shown that fixed, simple algorithms attempting a clearly suboptimal solution produced similar trajectories to the ones generated by the optimal techniques like Jordan's and Kawato's, but with a lot less work.

Many of these suboptimal solutions (e.g. Hogan ['80, '84, '84a, '85], Khatib ['85, '86], Flash ['85] and Hwang ['88]) are based on so called "potential field" or "impedance" methods. The potential field methods ultimately implement a nearly *reflexive* mechanism in that they eliminate search during trajectory planning, but they require a lot of computation to resolve the constraints imposed on the arm by the repellent potential fields of all the obstacles and the attractive fields of the targets. Furthermore, for this method to be effective, a complete and comprehensive world model must exist that models the locations of all obstacles. No means of model acquisition is proposed, and this is problematic, because that means the system either must operate in a benign *constructed* environment or else a complex search process must be used to acquire the world model that exists externally.

How then, do we resolve the problem of world modeling without an exhaustive search-based world model acquisition method?

Stopping far short of Brooks's outright rejection of representation, if we could construct a robust methodology that operates on incomplete information, this would reduce reliance on the world model. This methodology could be a layer in a more comprehensive subsumptive/hierarchical system. The intent is to model something akin to an animal's kinesthetic sense or the reflexive obstacle mapping world model of a blind person, which for obvious reasons, is incomplete.

We shall discuss a system that consists of interconnected CMAC elements, which can compute relaxed spatial trajectories and which uses a cerebellar habituation paradigm to adaptively learn to generate these specific trajectories given the right input context vectors. This system level is called ARTISTS, for "Adaptive Redundant Trajectory Information Storage System". A higher level layer of the system recognizes world-imposed constraints. Features corresponding to these constraints are embedded into the hyperspatial representation of the robot's kinematic coördinate system originally formed by ARTISTS. The higher level system is called ARTFORMS for "Adaptive Redundant Trajectory Formation System". This hyperspatial representation of analog features will largely replace the symbolic state of the world model used in symbolic AI systems which consists of discrete tokens embedded in an articulated representation (i.e. a digital database, blackboard, expert system, etc.). The result is a relaxed suboptimal search method.

The proposed method is similar to both MURPHY and the potential field methods. It is a weaker planner than MURPHY, and this is appropriate, because its role is to fill a less ambitious niche than MURPHY's. Unlike MURPHY, ARTFORMS is blind, but the ARTFORMS concept assumes that higher up in the hierarchy, there will be more powerful, perhaps sighted, planners that can present suggestions to ARTFORMS upon which it can habituate, that allow it to learn trajectory habits that may violate gradient descent rules. Gradient descent violation, i.e. moving the hand away from a target for a time, is sometimes required to get out of local potential "wells" in which potential field methods can get stuck. ARTFORMS also suffers from this problem without higher level help. However, given the property that ARTISTS has of forming habits, good or bad, it will tend to persist in habitual modes of behavior until a higher level system activity intervenes (by suggesting a violation of current rules) to change that behavior. The intervention described in chapter 3 is a punishment signal. In chapter 4, adaptive constraint satisfaction causes a behavioral change. Input for other changes, external to ARTFORMS is also allowed.

So we must view ARTFORMS as augmenting MURPHY in a fashion that should make MURPHY-like systems more computationally efficient and robust. ARTFORMS is, however, proposed as a complete replacement for the potential field methods, because it is more efficient. ARTFORMS is in a sense more powerful as well. Potential field methods settle into solutions that are a compromise among the potential fields of fixed attractive targets and repellent obstacles. These objects can move around, but the influences of the fields they generate fix the behavior of the system. ARTFORMS can adapt to obstacles and targets in a more flexible fashion. The obstacles have only local effects, so between local regions, there is more latitude in the selection of a solution. The challenge is to design this system in such a way that this additional flexibility, which may allow ARTFORMS to be able to recover from some of the potential well problems that thwart potential field methods does not become a detriment. The detriment that ARTFORMS risks is that in solving the problem in this fashion, residual redundancy is left in the problem. Controlling this redundancy so that *consistent* solutions result, while still allowing the freedom to select among trajectories, some of which are "dead ends", is a problem that is not *entirely* resolved in this dissertation. Resolution of these dead ends requires back track search, and is left to future work. Since the policies that are used to resolve the redundancies in regions not affected by the obstacles are based not on the obstacles and targets, but on rules that can easily be specified in the system of constraints and heuristics used in the system, fewer dead ends are likely.

## *1.3 Why This Thesis is an Improvement On Former Work*

Computational efficiency is the prime motivation for this work. The potential field methods and the "classical" closed form methods with constraint satisfaction are computationally burdensome and reliant on a relatively complete world model. This method suffers from neither detriment.

Since redundant arms have no unique transform in the inverse direction, from world to joint space, many path planners use "pseudo-inverse" techniques, that find matrix "inverses" to transform world space incremental movement vectors into joint space incremental vectors, given some disambiguating constraint. Two problems arise in this process. First, the pseudo-inverse is computationally expensive, being at best $O(d^3)$ complex[7], where $d$ is the square of the number of degrees of freedom. Secondly, it is not conservative, i.e., the transform of a "move" in world space to joint space and back to world space may diverge significantly (Klein ['83]).

### *1.3.1* *Analytical Methods*

Reducing the dimensionality of a redundant solution does not entirely solve difficulties encountered in iterative analytical solutions. Consider the forward kinematics of a simple 3 link arm like figure 1.1, page 8. Some direct solutions based on setting 2 or more joint increments equal and linearizing the differential kinematics might be used to seed an iterative solution. It would be unwise to do so without exercising considerable discretion, for reasons discussed below.

*1.3.1.1* ***Chaos in Newton Raphson Method.*** Recent results from Kramer ['92] indicate that simple iterative solution techniques like Newton-Raphson may not give good results, and in fact the method could be chaotic. This means that the algebraic approach could require the use of higher order "quasi-Newton" methods like Levenberg-Marquardt, conjugate gradient, etc., at the expected higher computational cost, and at the expense of having to know accurately what the mathematical model of the plant is. This is clearly unpleasant, especially for "long" kinematic chains! (And the problem is exacerbated seriously by extending the method to kinetics rather than just kinematics). In figure 1.3 a simple "2 sticks" problem is shown. The idea is to find the solution where the two endpoints are coincident. There are 2 solutions, one above and one below the "floor". Kramer's results summarized in *his* figure 7.5 show that in the (α,β) phase plane there are large regions over which very nearby initial

7 Which cost is in addition to the search cost for the world model, which is in general exponential in the size of the model.

conditions can result in different solutions. This notion of *postural switching* is discussed further in section 4.4.1.1. The interesting result is that just being very near a solution before iteration does not necessarily mean *that* solution will be the ultimate result. Chaotic results in a non-redundant problem like this one bode extremely ill for the prospects of a redundant solution using such a method. On the other hand, a simple, linearized pseudo-inverse may be a useful heuristic for ARTISTS.

### 1.3.2 ***Explicit Obstacle Avoidance is Essential***

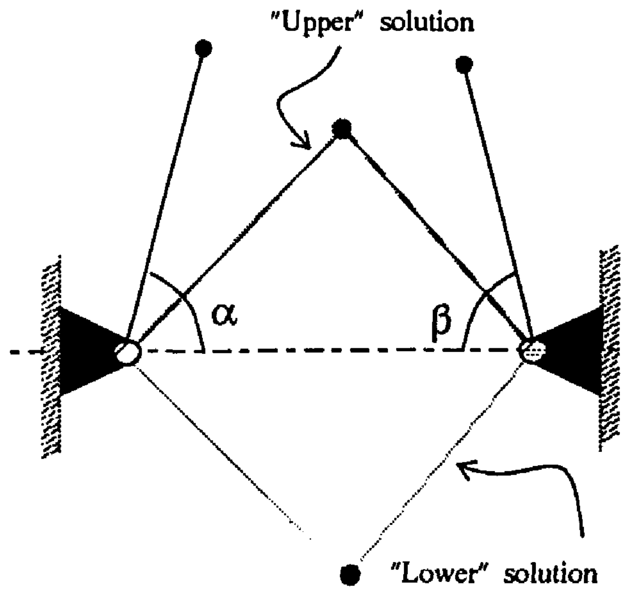


Figure 1.3: The 2 Sticks Problem

Kawato's method has no explicit obstacle acquisition method. Jordan's method is elegant and robust, but also lacks an explicit obstacle acquisition method.

### *1.3.3* *Computational Expense*

Kawato's method simply has too much time and space complexity to merit further discussion as a practical system. One *suspects* (in the absence of sure knowledge of his exact implementation) Jordan's method is also computationally expensive. Even if that were not so, there is still a front end price to be paid in his method that stems from a central thesis he holds, namely that direct inverse modeling does not work with supervised learning in redundant systems. (See sections 2.2.1 in addition to the next.)

### *1.3.4* *Jordan's Argument Against the Direct Inverse Method*

Jordan contends that redundant, or excess degree of freedom systems can not be trained effectively using the direct inverse modeling method of supervised learning for the following reason: if $x_1$ and $x_2$ are both inputs that produce a desired output y, then by repeatedly presenting the training pairs $(y,x_1)$ and $(y,x_2)$, to a neural network during training of that network, the network will eventually learn to produce a vector, $x$, for which $\sum(x-x_i)^2$ is minimized. Unfortunately, in this least mean square (LMS) derivation, $x$ is not, in general, a valid solution. Increasing goal directedness is Jordan's solution to the dilemma. (See section 2.3.)

*1.3.4.1* ***Linearity of Differential Inverse Kinematics.*** As Mel ['90] points out, though, inverse *differential* kinematics (used extensively in this treatise) is, by definition, linear. The discussion in Appendix C shows that this holds rather broadly, not just for infinitesimal increments. If the solution for a trajectory planner could be devised so as to exploit this linearity, the general argument against the LMS $x$ being a solution would not hold. Otherwise, the LMS solution fails for MLP networks as well as for CMAC networks. With MLPs, however, generalization is *a priori* arbitrary, and only becomes determined after the weights converge. Since the initializations of the weights are random, convergence to a final pattern may require extensive training, because the generalization inherent in the initialization of the weights may not fit the geometry of the problem. The result is that geometrically "dissimilar" inputs may generalize. This feature of MLPs is useful in cases where one might wish "dissimilar" inputs to generalize for some novel, and as yet undetermined reason, i.e., when the structure of the model being trained is not well known. In many control applications, it would be better to use a network with a known, deterministic, geometric mapping that dictates how generalization within the system's state space occurs. Further discussion of this topic is in section 2.6.

*1.3.4.2* ***"Reasonable" Generalization.*** Reasons for generalization among postures are easily determined for a manipulator. If two postures are close in a state space constructed from postural degrees of freedom (with a reasonable metric determining closeness) then they should generalize. If they are not close, they should not. (See figure 1.4).

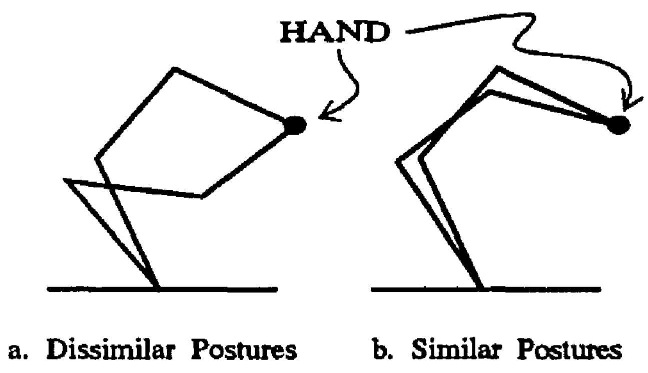


Figure 1.4: Postural Generalization.

By containing generalization to roughly coincide with differential regions around postures in state space, the linearity of inverse differential kinematics should favor convergence of trajectory formation.

This is because we thereby reduce the tendency to average non-linearly related inputs that are distant one from the other in the input space and yet produce very similar outputs. The "reasonableness" of the degree of generalization in ARTISTS is discussed in Appendix C, and compared to the results of section 4.8.6.

*1.3.4.3* ***Reasonable Input Mapping.*** If generalization occurs *only* locally within the state space, then the pair of positions in figure 1.4b should generalize, while the pair in figure 1.4a should not. It must be pointed out here that our postural mapping is not quite the same as the one usually chosen for manipulators, in that the absolute hand position is not part of the input vector. Instead, we shall use joint angles, together with the desired hand move. Details of this structured mapping, and the input and output vectors are described in section 2.4.2.

By modeling inverse Jacobian matrices rather than inverse static posture maps, the degree of redundancy is reduced[8], and together with the absence of the hand position from the input vector, this removes any geometric reason for the dissimilar postures of figure 1.4a to generalize, in spite of equal hand positions. Analytical means applied to this problem are well developed (Klein ['83]), but it is hoped that a neural network implementation could be considerably faster after training than the analytical method, and with a locally optimizing network, should be faster even during training with on-line learning engaged.

*1.3.4.4* ***Convergence Criteria.*** The postulated conditions for which supervised training of a direct inverse model network will converge on a consis-

8 Because we compute moves incrementally, not from one end of a trajectory to the other.

tent trajectory formation solution for excess as well as necessary degrees of freedom are: (1) on-line training, and (2) local generalization. The reason for the former is that on-line training during performance of trajectories favors goal directedness simply because only goal directed steps are presented as training exemplars. The reason for the latter is that local generalization largely prevents LMS averaging of highly non-linearly related trajectory step solutions.

### *1.3.5* *Keeping This Work In Perspective*

Keep in mind that I am not attempting to contravene Jordan's thesis, but am rather exploiting a "loophole" in his preconditions. The payback for this exploitation is considerable, namely the removal of the necessity of "pretraining" a forward model of the robot into an MLP before the model can even start to attack the problem of learning trajectories. My model trains on-line from the very start, thus acquiring the inverse model and trajectories simultaneously. (See section 2.2.1.) If a forward model is required, that too can be acquired incrementally and concurrently with the trajectory formation. (See section 6.3.)

This is a nice computational windfall, but an even deeper problem that it addresses is the avoidance of dependency on Jordan's (unavoidably) imperfect forward model. This method also allows the equivalent of retraining the forward model to account for changes in its physical properties over time. Jordan's forward model might be on-line retrainable, but based on considerable study of convergence and retrainability problems with MLPs, e.g. Fahlman ['90], this seems unlikely. Since Jordan's model relies on the perfection of his forward model, it is unlikely, if it were not comprehensively trained up front, that meaningful behavior would emerge from training in specific trajectories with on-line training of the forward model engaged. This argument is carried further in section 2.6.1.

The ultimate goal of this work is not just to implement robust efficient trajectory formation, but to include therein an inherent and robust obstacle avoidance methodology. This is developed in chapter 5.

Finally, Appendix A tries to put this work in context relative to neurophysiology, clearly a broad and speculative vista, but one which I argue portends certain advantages for the design engineer of the future.

### *1.3.6* ***Required Vindication of Results***

A necessary part of this thesis will be to demonstrate that the underlying storage method, ARTISTS, stably learns trajectories presented to it. Through many repeated simulations, it has been observed that this system, with and without obstacle avoidance, can reliably learn consistent, repeatable trajectories.

A systematic empirical search for adequate system parameters that guarantee success and the experimental results are discussed in section 4.8, page 105.

A rigorous convergence and stability proof using an analytical Lyapunov method is beyond the scope of this dissertation, but a discussion of a less rigorous "Lyapunov-like" argument is presented in section 4.9, page 136.

A convincing demonstration of ARTISTS converging on a stable learned set of trajectories taken from a pathologically redundant set of such trajectories, will be to demonstrate that this method handles adequately at least a redundant model similar to the one that Jordan proposed, and that it can learn similar trajectories.

#### *1.3.6.1* ***Computational Advantage Over Forward Modeling.***

Showing that ARTISTS converges on the trajectory ensemble just described is important because direct inverse modeling is more computationally efficient than forward modeling for reasons described in sections 1.3.3, 1.3.5 and 2.2.1.

# *Chapter II*

# *Issues*

## *2.1* *Supervised Learning*

The terms "supervised" and "unsupervised" learning are rather vague. It is unclear in the context of the current literature just where the boundary line lies between a supervised and an unsupervised learning paradigm. In some references, notably Mel ['87], unsupervised models are defined as ones not requiring an *intelligent* teacher. One would suppose this to be a human, but it might more generally be an "artificially intelligent" automaton or agent providing the input.

More demanding theorists, e.g. Rivest and Shapire ['87], would require that even a simple "black box" like the forward kinematic transform that provides the essence of the ARTISTS/ARTFORMS robot simulator is considered a teacher, and so let us call this model a supervised learning paradigm. We shall then reserve the nomenclature *unsupervised* learning for such paradigms as Kohonen maps, Grossberg's adaptive resonance systems, etc. However, an argument might arise that says such a *self organizing* system uses *itself* as teacher, and is thus supervised. This argument can clearly get out of hand!

Let us retain a rather strict definition. Supervised learning is the act of presenting to a system of equations, or other repository of information, such as a memory of the discrete or "fuzzy" variety, a set of $n$ vector pairs, $(\boldsymbol{v},\boldsymbol{t})_i$, where $\boldsymbol{v}$ is a context or input vector and $\boldsymbol{t}$ is a target or exemplar vector. The system is exercised to produce an output $\hat{\boldsymbol{t}}$, such that the error $\boldsymbol{t} - \hat{\boldsymbol{t}}$ can be computed. An adjustment of the parameters of the system that produced the output is then attempted to reduce the error term over a series of $n$ such vector pairs. The usual technique is to compute a gradient and descend along the gradient to reduce the squared error over the training set. Jordan ['88] gives a nice development of the general vector description of the method. Another development is shown here in section 4.6.4 (page 97).

## 2.2 Direct Inverse Modeling

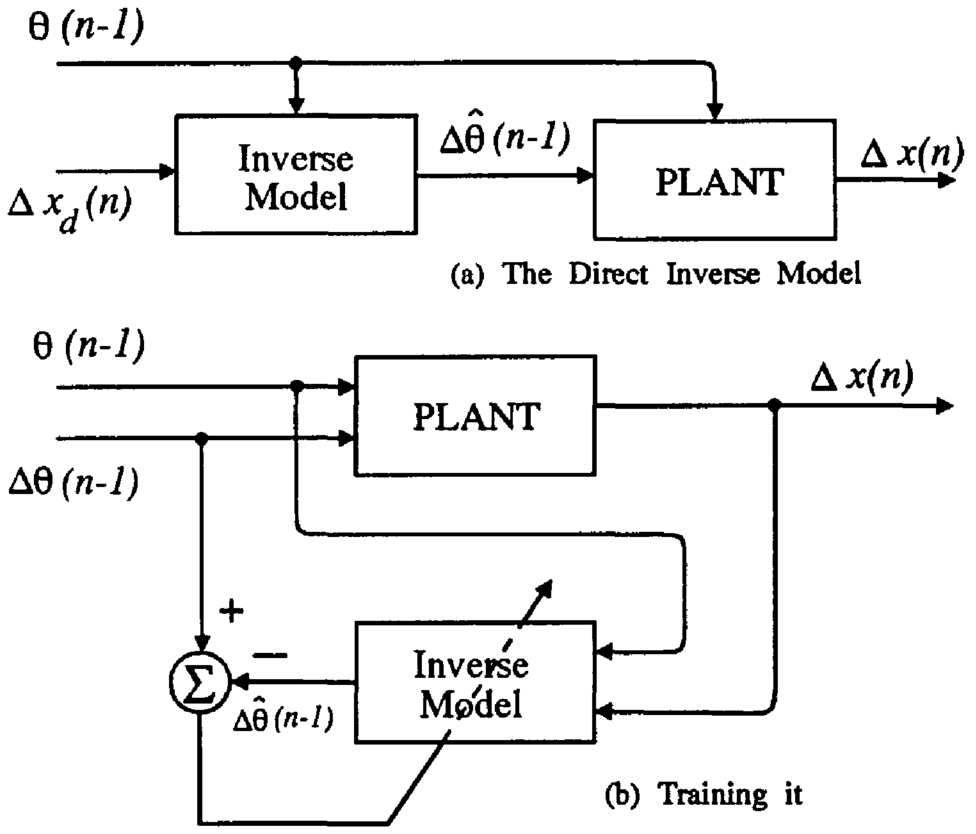


Figure 2.1: The Direct Inverse Method of Modeling

The following discussion is based largely on descriptions of direct inverse modeling versus forward modeling put forth by Jordan ['90].

Consider a mechanism, like a robot arm, which takes a joint position vector, $\theta$, and outputs a hand position vector, $x(\theta)$. If we cause the joints to move by

$\Delta\theta$, then we may rightly expect the hand to move by $\Delta x$. If we wish to move the hand (a typical robot task) by some amount, $\Delta x$, we must know what value of $\Delta\theta$ will cause such a hand move.

It would be nice to have a black box like the one labeled "Inverse Model" in figures 2.1 and 2.2 which could provide us with a reliable estimate, $\Delta\hat{\theta}$, for that desired input. Figure 2.1a shows a direct inverse system that has the intermediate value, $\Delta\hat{\theta}$, provided by just such a black box. Figure 2.1b shows how we would in practice train such a model. The dashed diagonal line indicates a gradient based adjustment of the model as is the case in other figures in this document. The important feature of this method is that the inverse model is trained (or adjusted) based on direct observations of actions of the plant. This direct observation method may be termed a world as model method.

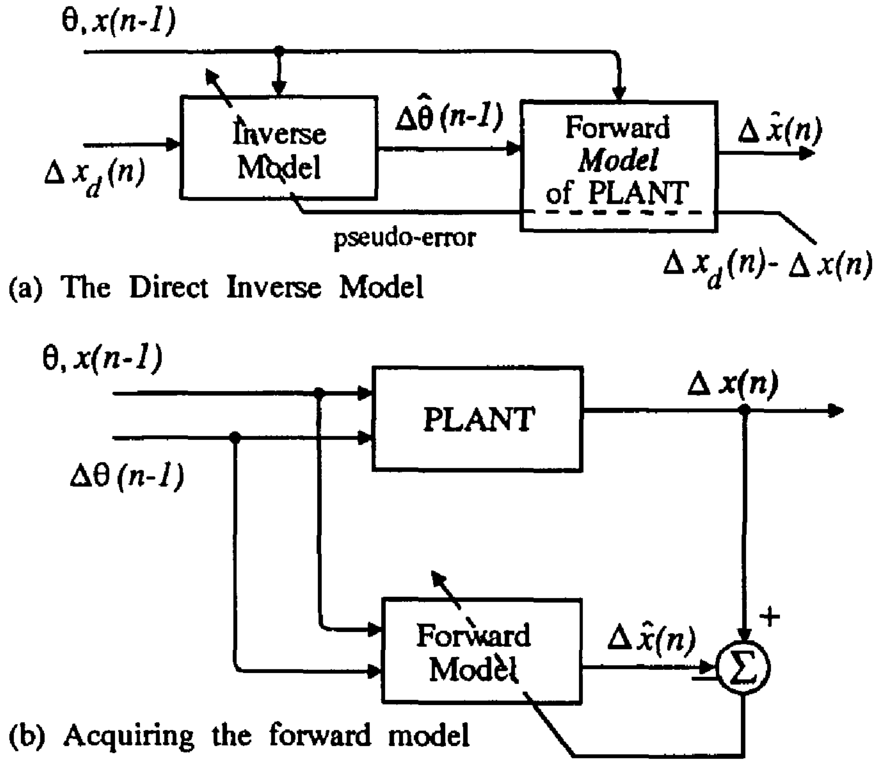


Figure 2.2: The Forward Modeling Approach.

### 2.2.1 *"Pre-training" a Forward Model*

In the case of forward modeling, a mathematical model of the forward kinematics of the plant must be acquired. This is done by exhaustively training a neural network from direct observations of random moves of the plant. This phase should be complete before training of the inverse model can ensue. This is the phase termed "pre-training" in this dissertation. During trajectory formation, the error observed between the actual plant moves, $\Delta x(n)$, and the desired moves, $\Delta x_d(n)$, is back propagated through the forward model, as indicated by the dashed line in figure 2.2a. This produces a "pseudo-error" signal that can be used to adjust the inverse model. For a thorough and understandable reference on back propagation. (See Wasserman ['88], chapter 3.)

The direct inverse model can be trained earlier in the process, based on direct observation of the plant, and so constructs the inverse model faster than the forward modeling method. However, Jordan argues that the forward modeling method is goal directed and as such can, via the gradient descent in figure 2.2a adjust the inverse model to solve for a particular solution out of a myriad of many-to-one solutions. In general, the direct inverse method can not find that solution, as was discussed in section 1.3.4.

In this dissertation we shall see, as mentioned in section 1.3.4.4, that local generalization and on-line learning capabilities of the CMAC allow a solution to be found via direct inverse modeling. We shall also discuss methods to force direct inverse modeling to be goal directed, both by the use of goal directed heuristics and by the use of constraint satisfaction. A discussion of the important aspects of CMAC are found in sections 2.5 and 2.6.

## *2.3* *Goal Directedness*

Direct Inverse modeling is not goal directed. What this really means is that it is opportunistic, in that it simply learns from what it observes. A goal directed method tends to force the system to behave in a manner that reliably reduces some goal oriented error metric. An example of such a metric is distance to target, direction to target or postural configuration, i.e. the relative magnitudes of an articulator's joint angles.

It is true that if a direct inverse system only trains randomly, without regard to any goal (as is the case in forward model "pre-training") there is no tendency to favor goal directed moves over goal divergent ones.

The direct experiential nature of direct inverse modeling assures that exemplars match the plant exactly; this should favor convergence on a representation that is a true plant model more rapidly than an indirect method (like forward modeling) could achieve. By contrast, indirect methods, though they can be crafted to maintain fidelity to goals, and may be better at adhering to goal directed policies, they may have desired rather than actual observed moves latched into the inputs during training. Thus these indirect methods will not only capture a true plant model more slowly, but in fact may risk attempting to train based on impossible actions (which is clearly not an efficient thing to do). If we can force strong goal directed policies that are efficiently computable upon the direct inverse method, an all around better solution is likely.

Chapter 3 will discuss the use of goal directed heuristics, and chapter 4 the use of constraint satisfaction to impose goal directedness on the system. In short, it will be argued that it is the use of constraints in Jordan's method that provides goal directedness to a larger extent than the use of forward modeling. Furthermore, if the goal directedness influences what the direct inverse model uses as training exemplars, then the same resultant success should be achievable by direct inverse modeling.

## *2.4 The Problem Statement*

The central problem this dissertation project addresses is that of how to control the trajectory generation of an arbitrarily long planar kinematic chain. The system is a simulation of a simple planar robotic arm, and the software is described in detail in Appendix F. The arm may be redundant or non-redundant. The object in each experiment is to start with an initial posture and generate a sequence of unit length hand moves that moves the hand from that posture to within a unit distance of a target. This action is called one *path segment.* More path segments are dictated by specifying multiple targets. The path segments can be *chained* by moving the hand along segments extending from target to target or they can be executed *radially* by always starting from the original (home) posture. The difference between the trajectories in these two cases became a pivotal concern in this study.

During execution of these experiments many conclusions were derived subjectively from observing the arm in motion and other conclusions were derived from studying the error metric data files the program wrote out during execution.

### 2.4.1 *A Brief Synopsis of the Software Solution*

More details are in Appendix F. An initialization file containing $x = y$ statements where $x$ is a variable name and $y$ the desired initial value is read in and interpretted by the simulator program. Over 50 system parameters can be adjusted in that fashion so experiments can be run without recompilation. Drawings of the arm are generated automatically in PostScript, and error data logged into a series of data files. These files contain logs of error metrics written out at the end of each "path segment". Up to 40 targets have been implemented in long sequences of experiments crafted by creating multiple initialization files and executing the program, "ARTFORMS", once for each initialization. These initialize and execute sequences were chained together in batch files, so that overnight runs could generate large collections of data for later analysis. These data were then used to determine adequate system parameterizations to meet the desired criteria of

- Paths that were nearly rectilinear in the work space.
- Minimal convergence time in epochs. (1 training epoch = 1 path segment).
- Minimal RMS error in postural constraints.
- Minimal CMAC memory saturation. (See section 4.8.2.7).
- Completion of the experiment (i.e. not getting stuck in unrecoverable postures).
- Avoiding mechanically disadvantageous postures.

### 2.4.2 *The Details of the ARTISTS Mapping*

ARTISTS is implemented to store trajectories for a 2D planar arm as shown in figure 2.3. We shall discuss the minimally redundant, 3 link, 3 joint case here, but experiments with other numbers of joints have been conducted.

The three joint angles are $\alpha$, $\beta$, and $\gamma$. The forward kinematic transform, $\mathcal{K}$ is straightforward[1],

$$x = T * \cos(\alpha) - H * \cos(\alpha+\beta) + F * \cos(\alpha+\beta+\gamma)$$

$$y = T * \sin(\alpha) - H * \sin(\alpha+\beta) + F * \sin(\alpha+\beta+\gamma).$$

No attempt to model the kinetic response of this robot arm was made. The kinematic response is suitably non linear, to demonstrate that CMAC is a sufficiently powerful computational model to solve a hard supervised learning problem. The extension to kinetics is a straightforward matter of increase of dimensionality.

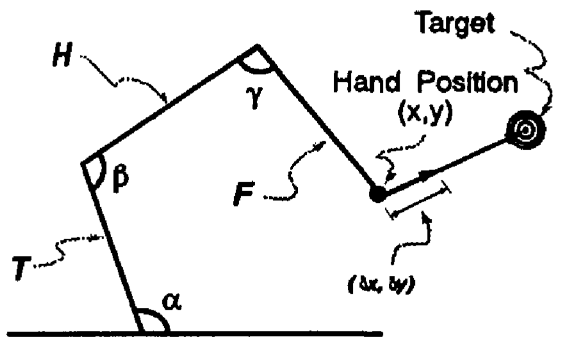


Figure 2.3: Simple Redundant Planar Articulator

A CMAC implements a mapping: $\mathcal{K}^{-1} : (\alpha,\beta,\gamma,\delta x,\delta y) \rightarrow (\delta\alpha,\delta\beta,\delta\gamma)$, where all are the obvious quantities, except ($\delta$x, $\delta$y), which will be treated as a unit vec-

1 With perhaps misguided anthropomorphic intent, the symbols were intended originally as F = forearm, H = humerus, and T = torso.

tor along a *desired* trajectory. So the input vector provides a desired straight-line trajectory direction, along with the current joint *posture* in order to excite a response, and the learning will train in the observed response at the *actual* observed trajectory step. Forcing the steps to be unit vectors results in an attempt at a constant velocity solution, and simplifies the input addresses for a more uniform input mapping -- only orientation information is contained in this 2 dimensional component. For obstacle constraints, the input vector might be changed to include 2 dimensions for path segment endpoint location. This could allow radically different policies to be used for forming trajectories aimed at 2 different target points along a straight line separated by an obstacle.

### *2.4.3 The Heuristic Criteria*

Let $\underline{S} = (\alpha,\beta,\gamma,\delta x,\delta y)$. The primary "weak gradient" heuristic rule is: If the CMAC returns $d\mathcal{K}^{-1}(\underline{S}) \approx (0,0,0)$, try a step, $\mathbf{R}$ (perhaps a random one), and observe the result, $\underline{S}^* = \mathcal{K}^{-1}(\underline{R})$. There are 2 possibilities: the step generated an $\underline{S}_{x,y} = (\delta x, \delta y)$ component such that the dot product $\underline{S}_{x,y} \cdot \underline{S}^*_{x,y} >= 0$. In that case, train on the step (i.e. train the CMAC at $\underline{S}^*$, not at $\underline{S}$), but not for the former case, because it moves the hand in the wrong direction.

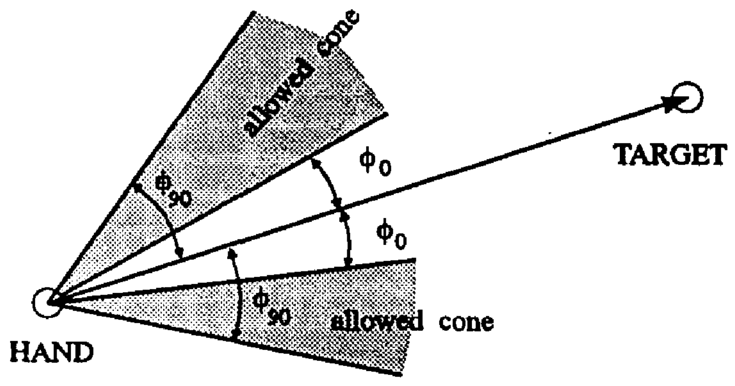


Figure 2.4: Gradient Descent Critic Criterion.

This first heuristic criterion accepts randomly generated heuristic steps only if they have between 0 and 90 degrees deviation from a straight line hand to target path. This heuristic allows the hand to deviate from the target but never move such that its distance to the target increases. A more general dot product critic algorithm is installed which can bracket acceptable steps by the generalized angles $\varphi_0$ and $\varphi_{90}$ as shown in figure 2.4. This more general criterion can variably constrain the hand moves (perhaps using a spatially distributed constraint parameterization) within the "allowed cones" shown in the figure. Note that this extension not only *allows* heuristic steps that diverge from the target, to allow back-track searching to find a path around an obstacle, but can actually *force* such heuristic steps by setting $\varphi_0 > 0$. Spatially distributed parameterizations are discussed in sections 3.3.1.2, 4.8.1.1 and A.1.1.1.

## 2.5 *Why Use CMACs For This Purpose?*

This section discusses both historical and theoretical reasons for using CMACs in the current development. Afterwards, section 2.6 contrasts and compares CMACs with an alternate and widely used type of neural network, the multi-layer perceptron (MLP), in an attempt to strengthen this argument.

### 2.5.1 *Historical Continuity*

During the past 6 years, the Robotics Laboratory at the University of New Hampshire (UNH) has conducted a series of simulations and real time studies probing the use of highly regularized neural network arrays called CMACs, for "Cerebellar Model Arithmetic Computer", inspired by the seminal work of James Albus ['79] and David Marr ['69]. CMACs have very successfully given our industrial robot controllers and simulators the ability to perform both repetitive and non-repetitive actions defined by desired actions in the sensor space, while requiring only qualitative knowledge of kinematics or dynamics.

No *exhaustive* description of CMACs will be included here; a comprehensive tutorial and descriptive article is Miller ['90d]. Of course Albus ['81] is also good, but not as germane, given the specificity of the former to the problem at hand. Important discussions of generalization, quantization and memory size are included in section 2.6.2.2 and in the comparative discussion of generalization for MLPs and CMACs in section 2.6.2. The CMACs used in this series of experiments vary from Albus's original design in two major ways; the CMACs use: (1) linearly tapered receptive fields, and (2) An's optimal receptive field distribution. Both these aspects were covered in An ['91]. Some experiments using rectangular fields were tried and though they may have been adequate for the system, the linear tapered field model worked better. No further comprehensive comparative study was engaged outside this subjective assessment. An's arrangement for 2D inputs is shown in figure 2.7, page 43.

### *2.5.2* ***The SERVO Level Controller***

To date, the Robot Lab's robot control systems have contained a crude fixed gain linear controller. This controller was completely insensitive to the base line or time dependent dynamic parameters of the system, and thus was incapable of generating anything other than very crude, clumsy and inaccurate moves of the articulator arms under control. Adaptive CMAC modules, operating in parallel with the fixed gain controller in both the feed-forward and feedback paths, allowed the system to learn the highly non-linear error function that emerged when the actual and desired trajectories were compared. The results were gratifying and, as the series progressed, the system was deemed completely sufficient

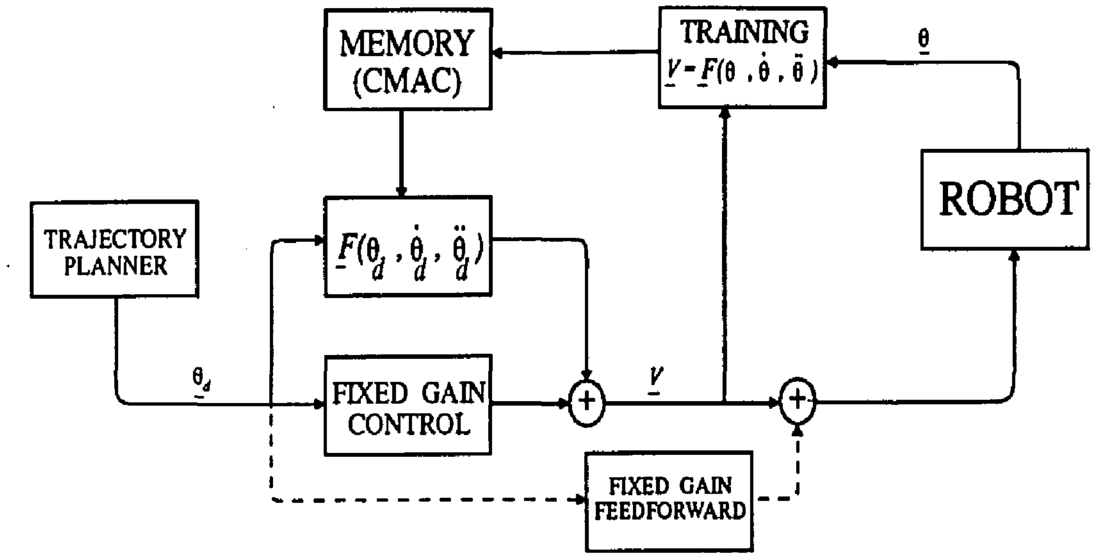


Figure 2.5: Simplified Diagram of UNH SERVO Controller

for the job of on-line real-time adaptation of these brain stem (SERVO) functions in the absence of an analytical model of the robot dynamics (Miller, *et al* ['86 - '90e]), but the problems were posed using only non-redundant[2] mechanisms.

*2.5.2.1* ***Converting Feedback to Feedforward Anticipation.*** In spite of the lack of plant redundancy, the relevant feature of the Miller/Glanz/Kraft controller is that it effectively turned feedback control of a non-linear plant into a largely feed-forward operation; (see figure 2.5). The existence of long time delays in biological systems has forced nature to develop just such strategies as this. There are similarities between this method and the ones Houk ['90] describes. He describes a kind of evolutionary adaptation called a *quasi-feedforward* process. In a *quasi-feedforward* process there is only limited reliance on feedback, much of control being performed in a feedforward fashion. After failure, the feedback mechanism he characterizes as being much like back-propagation adjusts parameters retrospectively. Without the feedforward

2 See section 1.1.2, "Varieties of Redundancy".

activation of this method, feedback loops controlling non-linear plants with long time delays could go unstable.

It may be argued that electronic systems are fast enough to have outgrown nature's requirement here. The counter to this argument is that with the addition of synthetic vision and heuristic back-track search in path planning, dictated by the existence of redundant mechanisms operating in the presence of obstacles, such time delays in fact do occur. So *this* proposed method strives to capture this reduction of reliance on feedback in the conversion of feedback of observed movements into a feedforward process of movement *anticipation* during path planning.

### *2.5.3* ***The Existence of CMAC Hardware***

In a recent development, hardware implementations of CMACs have become a reality (Miller ['90b, '90c]). The prototype hardware design developed at UNH has been commercially developed by Klein Associates of Salem, New Hampshire. The commercial systems provide sufficient speed (>1 KHz) and storage capacity (1 Mbyte) for most anticipated problems in robotic control.

### *2.5.4* ***Multiple Interacting Neural Networks***

The obstacle avoidance implemented in this study involves no explicit world model. It requires simply that CMACs implemented in parallel learn by experience the inverse kinematic transform of a simulated robot arm with biases embedded in it to represent obstacles in a coarse fashion. The outputs of these CMACs blend together to generate the system's control signal, a string of elemental move commands to be passed down to a servo level controller (like the Miller/Glanz/Kraft controller, Miller *et al* ['86 - '90], shown schematically in figure 2.5), which then tracks the path. The networks here must be roughly equal in timeliness, so fast convergence is essential.

In ARTFORMS-1, reinforcement learning[3] is used to train a short term memory (STM) of what *not* to do in certain contexts irrespective of why not to do it, but the locality around this context point in state space *is* important. When an obstacle is encountered, the system receives a non-specific diffuse punishment signal, called SLAP, which engages training in the STM whose output serves as a bias to the output of the long term memory (LTM) and ultimately becomes part of the training signal for the LTM. The LTM contains *an* inverse[4] kinematic (i.e. inverse Jacobian) mapping for the arm.

ARTFORMS-2 does something similar, by using a constraint satisfaction paradigm, which is moderated by a STM projection that can alter the constraint equations in a meaningful way.

The STM module in either case must have a very high learning rate[5], so it captures its data quickly, and in the absence of an obstacle-indicative input it decays by being trained using a zero valued exemplar vector, and a lower learning rate.

The LTM does not have a decay process. It simply has a continuous dialog presented to it, consisting of:

- the current context (current position and target position),

3 See section 3.4, "Reinforcement Learning".

4 It should be kept in mind that there *is no inverse model* of the arm kinematics because of postural redundancy, but the proposed mapping provides a non-unique mapping from desired kinematic response to the joint perturbations that cause such a response, and so is "like" an inverse model.

5 Learning rate, denoted by the symbol, $\eta$, determines how fast a neural network converges on a solution. In other iterative methods, this might be termed "step size". Its value can range from 0 to 1.0.

- what is commanded *including* the STM bias and
- the resultant action (feed back of the observed response of the system one time step in the future).

So whatever moves the robot makes, for whatever reasons, are simply observed and trained into a hyperspatial representation of the robot's kinematics. If such an "observe and mimic" methodology can be implemented and shown to be a stable repository of trajectories, it will implement a localized motor program "habituation" paradigm. It has been postulated by Albus, Marr, and Houk that this is the function of the cerebellum. This forms one argument, albeit metaphorical, for the use of CMACs to implement the method. More practically however, the properties of local generalization and fast on-line training are the underlying reasons. Habituation is discussed in greater detail in section 3.3.1.1, on page 65.

## *2.6* *CMACs versus MLPs*

It is true that many control systems and trajectory planners have been implemented with multi-layer perceptrons. There are some serious concerns related to the use of MLPs in the current context that are outlined in the comparison of CMACs and MLPs found in this section. As with CMACs, detailed descriptive treatment of MLPs is omitted. Many descriptions are available, e.g. Wasserman ['88].

### *2.6.1* *Some Worrisome Properties of MLPs*

My concern for the feasibility of a less than exhaustively trained MLP used as a forward model stems from the manner in which an MLP solves a "surface-fitting" problem versus the way in which a CMAC does.

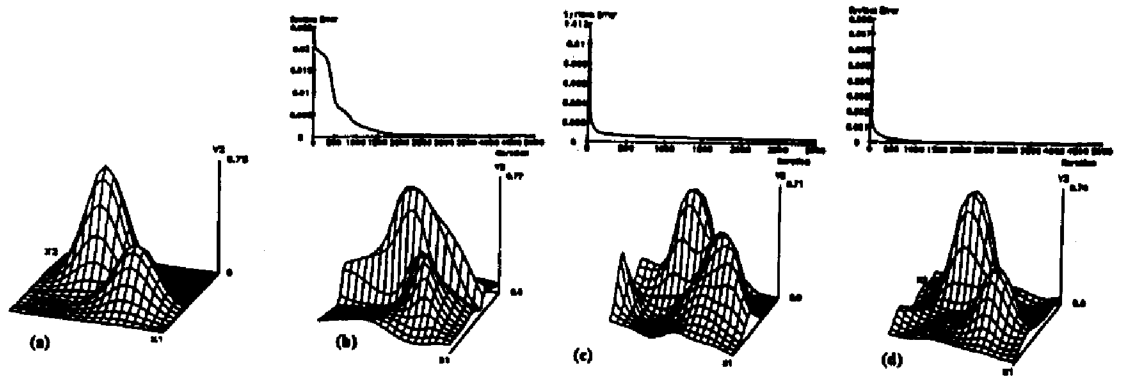


Figure 2. Comparative evaluation of learning rates and approximation capabilities: (a) surface to be learned, consisting of two Gaussians; (b) back-propagation net with 48 nodes in the hidden layer; (c) functional-link flat net with 48 enhancement nodes; (d) functional-link flat net with 200 enhancement nodes. Learning time in (d) is 0.14 that of (a).

Figure 2.6: How an MLP Converges
Copyright (C) 1992, IEEE Computer.

A CMAC solves a problem quickly everywhere in state space that it visits and leaves a very low level residual error everywhere. See figure 3.5, on page 69 and contrast it with figure 2.6. The latter result, reprinted with permission[6] from Pao and Takefuji ['92], shows the readout of a network called a functional link network, which is one of many variations of backpropagated MLPs, and is one that has an order of magnitude faster convergence rate than standard backpropagation. Even so, there are regions wherein the network approximation is highly accurate and other regions where it is quite inaccurate. So testing the robustness of such a system by applying uniform low level noise everywhere, as

6 Copyright IEEE Computer.

Jordan ['88] did, does not really model the inherent deficiencies of the MLP as system model.

An MLP can not hope to be a suitable fit for a problem unless it has "sufficiently" many nodes with the "right" connections. For the *general* case this means *many* nodes. For the *specific* case it means custom tailored MLPs to fit the architecture, i.e., just the sort of thing we are trying to avoid by moving away from purely analytical methods.

Contrarily, robustness in the presence of low level noise *is* consonant with the manner in which a CMAC tends to solve a problem and arguments presented in section 4.8.7 tell of how CMACs in *this* architecture contribute to an innate robustness. Memory consumption and inherent noise are directly related. An upper bound on the worst case inherent noise a CMAC imposes on its host system are discussed in Appendix B, and results in section 4.8.7 indicate a wide range of system design latitude to allow an effective tradeoff between system performance and memory consumption.

### *2.6.2* ***Local and Global Generalization in MLPs and CMACs***

The concept of generalization can be stated from two vantage points.

- The connectionist's definition: The degree to which a neural network learns about novel situations from familiar ones, i.e. ones upon which it has been trained. In other words, if a supervised learning system is presented with exemplar $x$, how well can it predict a response to exemplar $x'$, which is purported to be "similar".
- The localist's definition: The degree to which the receptive fields of local basis function networks can overlap. With a radial basis function, this quantity is expressed as the radius

of the receptive fields. With a CMAC, it is expressed as an integer quantity representing the number of discrete state space points in a receptive field, corresponding to weights stored in memory locations.

*2.6.2.1* ***Deterministic CMAC Generalization.*** For a CMAC, these two definitions are more or less equivalent. If there is a large receptive field (i.e. many weights) then other receptive fields can share many or almost none of these weights. The number of weights in a CMAC receptive field is, by convention, represented by the variable, C. Since the state space points are widely and uniformly distributed throughout the state space, it is possible then for widely separated inputs to overlap, and thus generalize. If there is a small receptive field, the probability of two fields sharing weights is small, unless the inputs corresponding to the two receptive fields are very close together, hence only *very* similar inputs share weights, and thus little generalization occurs. In fact if $C=1$, we have table lookup and *no* generalization occurs. For the simple case of a one dimensional CMAC, the effect of broader or narrower generalization can be seen in figure 2.7, which shows how differing values of the generalization parameter might affect function approximation of a sinusoid. Segee ['92] discussed how the width and profile of the receptive field affects learning speed as a function of the spatial frequency of the exemplar function. This result together with the observations of Appendix D will connect the methods of ARTISTS/ARTFORMS with previous traditional methods of non-linear control. See section 6.2.

*2.6.2.2* ***Generalization and Quantization in CMAC.*** The term codon representation is found in Marr ['69]. It refers to the degree of coarseness of the input vector coding. Coarse coding is a term also used in reference to CMACs. The input vector must be discretized for a CMAC, and so scaled inte-

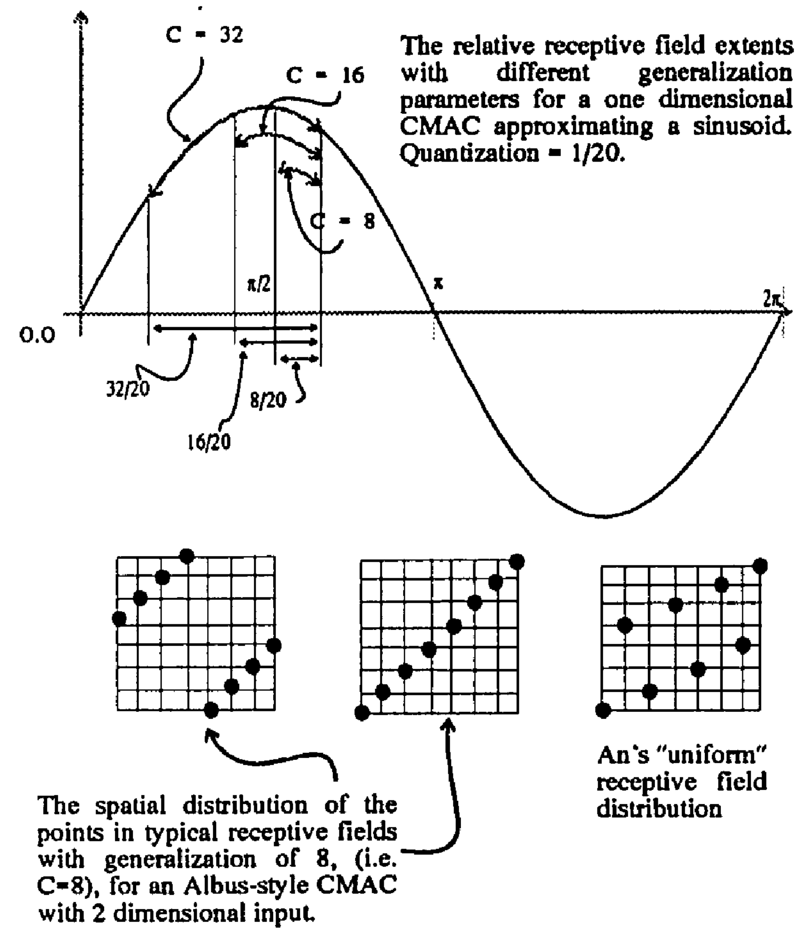

C = 32
C = 16
The relative receptive field extents with different generalization parameters for a one dimensional CMAC approximating a sinusoid. Quantization = 1/20.
C = 8
π/2
π
2π
0.0
32/20
8/20
16/20
The spatial distribution of the points in typical receptive fields with generalization of 8, (i.e. C=8), for an Albus-style CMAC with 2 dimensional input.
An's "uniform" receptive field distribution

Figure 2.7: Generalization and Receptive Field Extent

gers are used. The scaling can be as high resolution as necessary to assure an accurate simulation model. But if the resolution is *very* fine, the CMAC virtual memory space can become unworkably large. Since a change of 1 unit in this representation does not necessarily result in even a measurable change in the output of the CMAC, it is very wasteful to leave matters thusly. In order to reduce memory consumption, the CMAC can further discretize an input vector to the degree that a single unit increment or decrement does produce a measurable output change (on average). This discretization control can be exerted in UNH_CMAC by using an array called qnt_vec[ ]. This vector contains in each corresponding component the number of units change in the input vector component that will ensure a new weight is enlisted in (and thus an old one dropped from) the receptive field. This allows a simple adjustment on a per coordinate basis of discretization that is independent of the plant simulation discretization. One unit of change at this coarse level is called a codon unit[7]. In figure 2.7, for example, each codon unit is $\frac{1}{20}$ linear units (*lu*). If the plant were discretized with 1 bit representing 0.01 *lu* then a codon unit would be 5 plant simulation quanta.

*2.6.2.3* ***Non-deterministic Generalization in an MLP.*** Multi-lay- er perceptrons, on the other hand, can exhibit very broad and almost uncontrollable generalization. Figure 2.8 shows a typical 3 layer multi-layer percep-

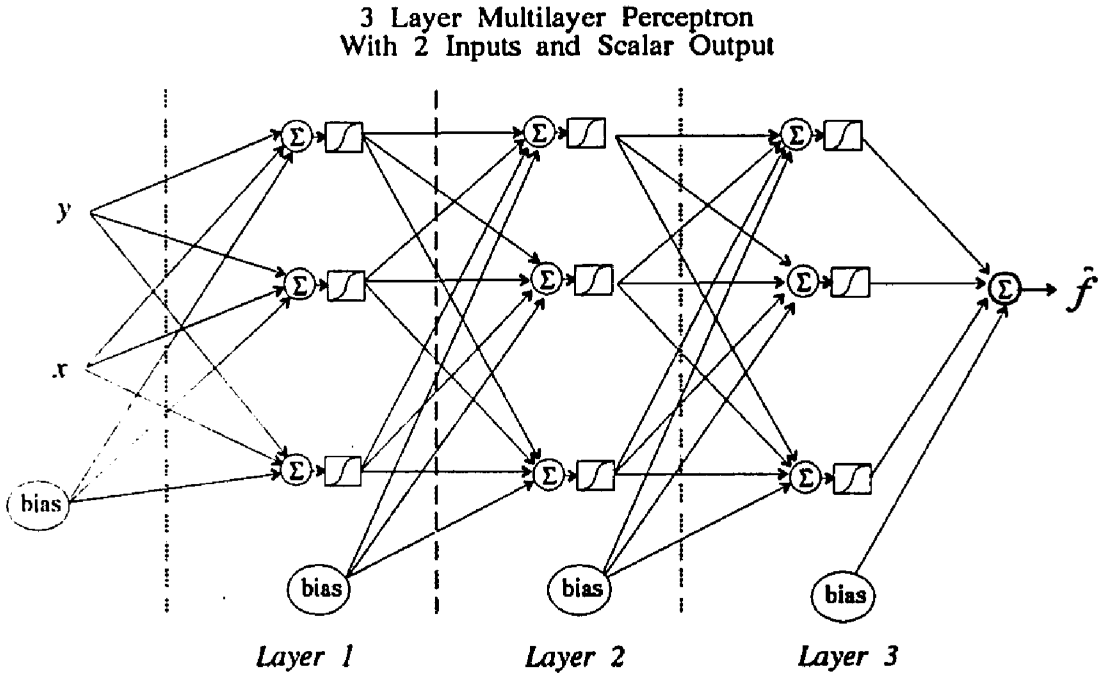
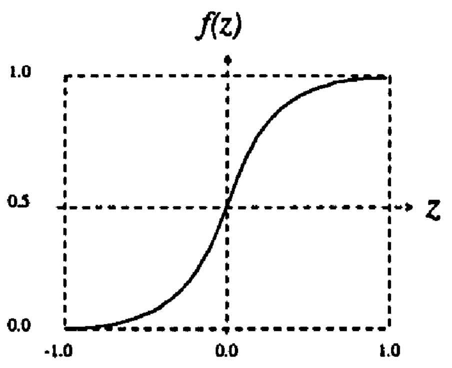

3 Layer Multilayer Perceptron
With 2 Inputs and Scalar Output
y
x
bias
bias
bias
bias
Σ
$\hat{f}$
Layer 1
Layer 2
Layer 3

Figure 2.8: A 3 Layer Multi-Layer Perceptron

7 After Marr ['69].

$$f(z) = \frac{1}{1 + e^{\frac{-(z+b)}{T}}}$$



$$z = \sum_i w_i x_i$$

Commonly used as the "non-linearity" in multi-layer perceptrons, this sigmoid is drawn for T = 0.2 and the bias, b = 0.0. z is just a weighted sum of the inputs. Training is accomplished by adjustment of the weights using backpropagation to solve the credit assignment problem related to the weights. This unit will be sensitive to large values of z, so if the weights attached to its inputs are all large, this unit will be saturated in an ON state, even for modest input values. Since its derivative is also small there, its weights will not likely change in response to training exemplars that drive its inputs high. This can be interpreted as a kind of local generalization.

Figure 2.9: The Sigmoid Non-linearity

tron. The sigmoid non-linearity of figure 2.9 is the heart of an MLP's ability to approximate functions or act as a pattern classifier. These two capabilities are in a sense equivalent operations. If a network can approximate a function, then the surface represented by that function will have closed features (i.e. simple humps or depressions) or open features (i.e. ridges or valleys) which can be intersected by hyperplanes to form boundaries of respectively closed or open decision regions. Unfortunately for an MLP, the network size and interconnectivity determine the kinds of functions (or decision region sets) a particular network is capable of computing.

*2.6.2.4* ***The MLP as General Function Approximator.*** Hornik, Stinchcombe and White ['89] showed conclusively that a 3 layer MLP is capable of approximating any Borel measurable function from n-space to m-space, for any n,m natural numbers. This class encompasses virtually all useful functions encountered in engineering applications. They did not however define a method of training such a network. In other words, figure 2.8, given enough units in each layer, can be a general function approximator. But getting it to adjust to approximate that function is not a well defined procedure. Their argument was more general in that it allowed other than sigmoid non-linearities, but constraining the discussion to just the sigmoid class of networks, their argument can be summarized as follows.

First, observe that one node from figure 2.8 is able to separate a hyper-space into two half-spaces, because the input function is just the dot product of two vectors, the input vector and the weight vector, the latter of which is the normal vector to the dividing hyper-plane. The transition between the two halves of the divided space can be as abrupt or as gentle as you please by adjust-

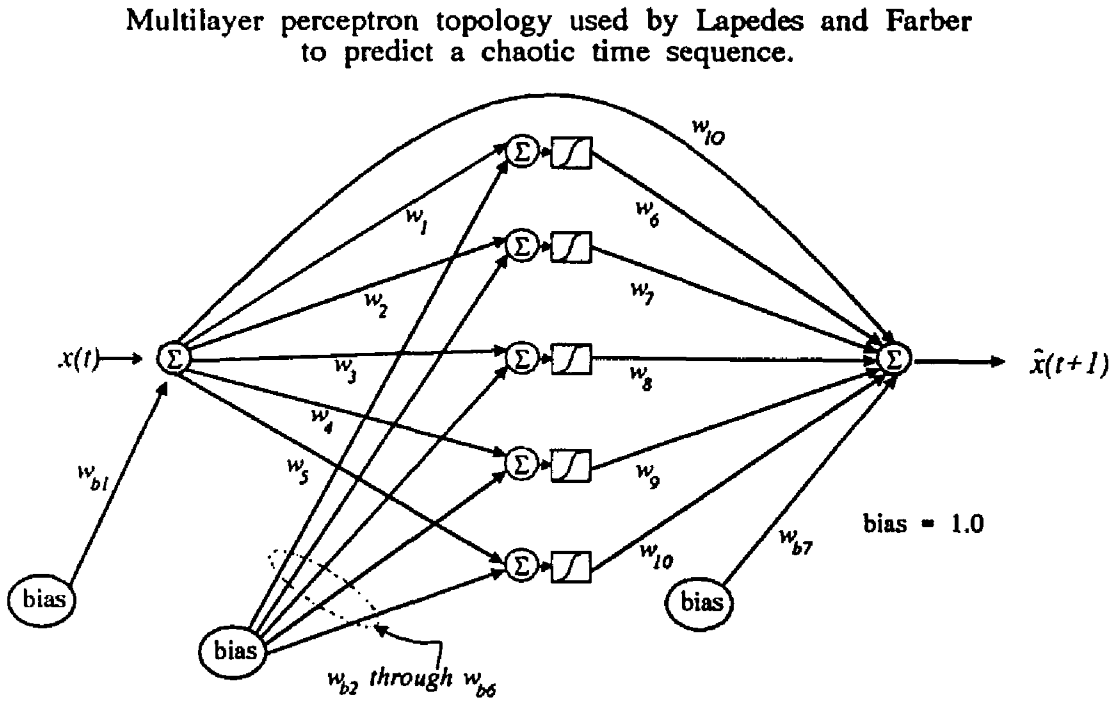


Figure 2.10: Lapedes and Farber's Chaotic Sequence Predictor Network

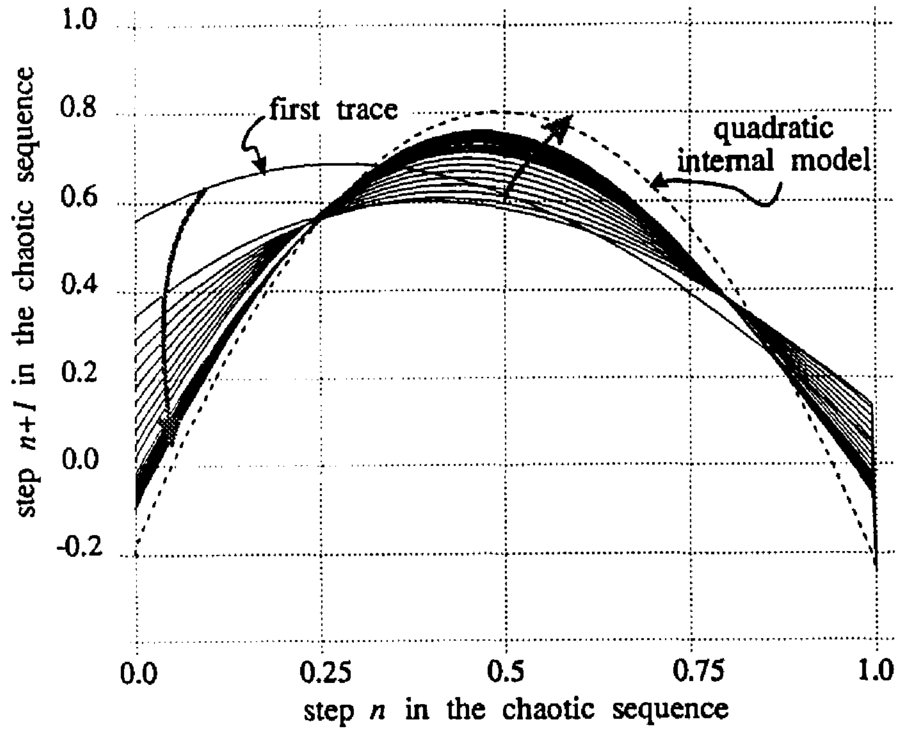
ing the slope of the sigmoid, by changing the parameter T of figure 2.9. So the orientation of the separation plane is defined by the weight vector and its displacement from the center of the space by the bias weight. If there are two such units in the same layer, they can project onto a single unit in the second layer which can, by conjoining the outputs of the two units in the first layer divide the space into 2 open regions. If the first layer has 3 such units, only then is it possible for the second layer target unit to *enclose* a convex region in the input space, or approximate a surface with one "bump" on it. In fact more and more units in the first layer all projecting onto a single unit in the second layer can define any convex region you please. Suppose there are multiple such groups of units in the first layer, and each group projects onto one unit in the second layer. Any connected region (a conjoint of convex regions), or arbitrary shaped bump, or any disjoint set of convex regions (or collection of simple convex bumps) but not both, can be approximated by the target node in the second layer. The third layer can then conjoin and disjoin regions (features) output by the second layer, which is equivalent to computing any arbitrarily complicated surface or function. And furthermore, backpropagation or a variant thereof can (hopefully) train it (eventually).

It is these two parenthesized conditionals that pose the problem. We are faced with the prospect that there is just no good way ahead of time to decide how to size or connect such a network in order to perform a given job without undergoing a thorough analysis of the nature of the function in hand, in which case it is probably not necessary to use a neural network to perform the computation. So what is really needed is a generalized three layer network, which can reliably map *arbitrary* functions without exhaustive *a priori* analysis of the function, which may not be a known quantity. Throughout this dissertation, that is precisely the role in which CMAC is cast.

*2.6.2.5* ***MLP and CMAC Equivalence.*** A CMAC can be shown to be equivalent to a three layer multi-layer perceptron, in which the first two layers are hardwired (i.e., have fixed interconnection weights). These first two layers define the topology of the function the network is capable of approximating, and are called hidden layers. A major problem with an MLP is that it must perform two jobs simultaneously: train the hidden layers to understand the space, and train the output layer to understand the function. All the hidden layers should be doing is dividing the input space up into local compartments. The output layer can then conjoin or disjoin these to form arbitrary sets of convex regions, and then weight these component regions to finish defining the output. The equivalent of a CMAC could be built using a three layer MLP if a nearly infinite number of sigmoid units could be connected together in a regular pattern such that groups of these units could each subtend limited regions of the input space. Each of these little subnets would then fulfill the role of a CMAC receptive field. Fahlman ['90] attempted to do just such a thing adaptively in his cascade correlation architecture.

#### *2.6.2.6* ***Local and Global Generalization in a One Dimensional Problem.***

Lapedes and Farber ['89] wrote a paper on chaotic time sequence prediction. Their network simply learned the internal representation of chaotic sequence generator which operated by feeding back the output of a quadratic logistic function as its next input. By cycling this generator repeatedly, an apparently random sequence resulted. If a network observes the sequence of inputs and outputs of the generator and trains in supervised learning fashion, all



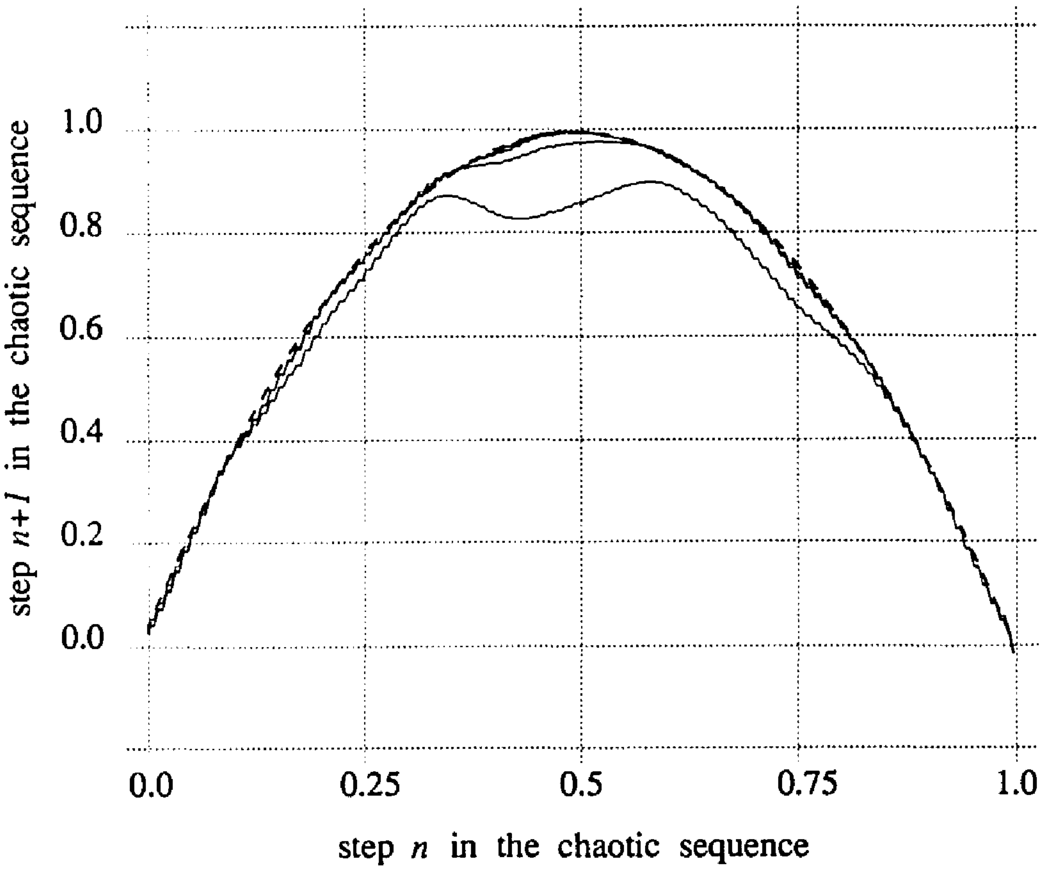
Lapedes and Farber's chaotic time series experiment duplicated. The light gray arrows show the approximate direction of convergence of the backpropagation algorithm. This rather rapidly convergent experiment is a fast learner principally because it has a carefully chosen architecture, and weight initialization. In other words, it "fits the problem". Other initializations proved to be more than two orders of magnitude slower even with the same network topology.

*MS Error Per Epoch*

| | | | |
|---|---|---|---|
| *17.898361* | *6.207622* | *2.366761* | *1.260756* |
| *11.590284* | *5.205238* | *1.932657* | *1.135156* |
| *9.934926* | *4.324663* | *1.732864* | *1.023141* |
| *8.574083* | *3.562446* | *1.557949* | *0.923478* |
| *7.331949* | *2.912544* | *1.401226* | *0.834997* |

Figure 2.11: Convergence of a Sequence Predictor with an MLP

it is really doing is learning a quadratic. Such a function has one smooth hump, and so the topology of figure 2.10 should suffice. I used an MLP of that topology and standard backpropagation, with a heuristic cycling of the learning rate, to attempt to learn the function as Lapedes and Farber had done. The results are seen in figure 2.11. What I discovered in the process of this effort was that the weight initialization was critical. If I initialized the weights as large weights with a wide variance (weights range from +/-5.0) the result was figure 2.11. When I used the "conventional wisdom" of small magnitude random weights



*MS Error per Epoch:*

| | |
|---|---|
| *0.581080* | *0.019997* |
| *0.035263* | *0.019170* |
| *0.022182* | |

Figure 2.12: Convergence of a Sequence Predictor with a CMAC

(ranging from +/-0.5) the result was that the function converged several hundred times more slowly, and the original approximation (similar to the first trace of figure 2.11) persisted for hundreds of epochs with only minor changes in downward concavity to try and approximate the quadratic. Why might this be? I contend that it is due to overgeneralizing during training.

If all the weights of an MLP are similar and small, then on average all the sigmoid units attached to the summers will tend to be presented with weighted summed inputs that are near the centers of the sigmoid functions. That happens to be where each sigmoid has maximal slope, so gradient descent changes every unit on every training step. Fahlman ['90] calls this situation *herd effect.* This effect is a kind of global generalization: a system-wide response to a single gradient adjustment. Furthermore, if most weights are similar and small, then it is probable that *many or all* of the units will tend to respond strongly to a particular input, which is a form of global generalization: a system-wide response to a single input vector. Both these global generalization affects are problematic, in that they slow learning down. If the initial weights are set with a wider variance, however, then there is a higher probability that a *limited* subset of the units will respond strongly to a particular input, while others will ignore it due to having sigmoid inputs negative. If these two subnets of sigmoids units are driven into the saturation region, they will persist there in spite of training adjustments due to having near zero slopes. This represents a form of localization, or the emergence of local generalization in the MLP network, albeit a rather haphazard one. My final weights and Lapedes and Farber's both exhibited a rather broad variance.

A further slowdown of learning is caused by the fact that MLP weights are not changed after every iteration because so called *incremental learning* tends not to work well for MLPs. Rather, *batch learning* is typically used, wherein the errors encountered in the training steps are summed throughout an entire epoch of training exemplars, and the weights are all adjusted at the end. The need for this is due in part to global generalization. CMACs on the other hand tolerate incremental learning well due to their innate localization capabilities, and this is essential to on-line learning, which requires that the system learn at every iterative step. The speed of convergence of CMAC versus MLP is dramatically demonstrated by a comparison of figures 2.12 and 2.11 wherein it can be noted that CMAC accomplishes in 5 epochs a degree of accuracy of function approximation that will take the MLP almost 500 epochs to match, and this is with a carefully crafted MLP with a particular weight initialization. With a less serendipitous initialization, the MLP's learning rate slows down by up to 2 orders of magnitude.

So in summary, the promise that "similar inputs generate similar outputs" that is mistakenly attributed to MLPs is in fact the main strength of radial basis functions, sparse distributed memories and CMACs, i.e. the set of *local basis networks*. MLPs "generate similar inputs" under rather ill-defined conditions.

*2.6.2.7* ***CMAC-like MLPs.*** Extensions to the cascade correlation architecture (a variant of the generic MLP postulated by Fahlman ['90]) can be devised that can adaptively "grow" *state space detectors*, or locally receptive fields, similar to those inherent in the CMAC architecture. But this does not improve the time complexity problem for the MLP which still requires a complete forward activation of the network. This can be so costly that if the problem gets harder than the simple one posed in Lapedes and Farber, notably with higher dimensional representations, a massively parallel computer would be essential for a reasonable implementation. The MLP is exponential in the dimensionality of the problem and number of layers while CMAC has a computational complexity that is *linear* in the generalization parameter, C, and dimensionality. It should be remembered that, if the MLP is well fit to the problem, as a modified cascade correlation architecture could be, and in the end could produce something very much like the CMAC architecture, it would have some properties like smoothness of function representation that CMAC lacks. So here is grist for a tradeoff decision, where the detriment of computational complexity must be weighed against the need for smooth function approximation. For control applications, where immediacy of response is often more important than accuracy, and as is often quoted, that "sign is more important than value", it should be apparent that CMAC will usually win in this tradeoff decision.

## 2.7 Conclusion: CMACs Are Appropriate

The results of sections 2.5 and 2.6 are summarized as follows: Since no neural model except CMAC exhibits the necessary speed of convergence to do on-line incremental learning (a requirement for real-time adaptation), and since only local basis function networks (of which CMAC is a kind) exhibit the necessary localization property, and given the milieu of CMAC activity and experience in the Robotics Lab, it makes sense to study how these CMAC modules might be used to implement path *planning* actions. The goal is obviously to provide a front end trajectory planner for SERVO trackers like figure 2.5 (page 36). A *redundant* arm will be used (for its obstacle avoidance capability). No adjunct fixed gain linear controller will be implemented to provided the CMACs with guidance as in the experiments of Miller *et al*; since generalized path planning has no underlying linear model, it may not be very helpful to do so. The goal will be plan trajectories for an unknown robot configuration, given only the ability to observe the joint angles and hand position.

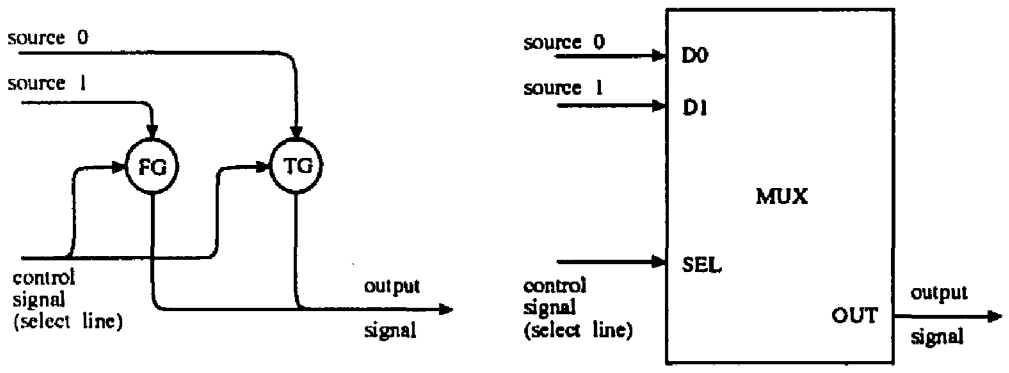
# *Chapter III*

# *The First Generation Solution*

In this chapter, a trajectory planner is developed that exploits an erroneous assumption, but nonetheless involves basic concepts that become the basis for the successful implementation of chapter 4.

## *3.1* *A Preview*

In addition to this preview, the reader may find section 3.4.1 instructive. In that section, a very much simplified system called 2DTFORMS is introduced that is analogous to ARTFORMS, but is of lower dimensionality and hence is easier to understand.



(a) A typical dataflow diagram "multiplexer"

(b) A simplified dataflow multiplexer based on circuit element symbol

Figure 3.1: A Dataflow Multiplexer

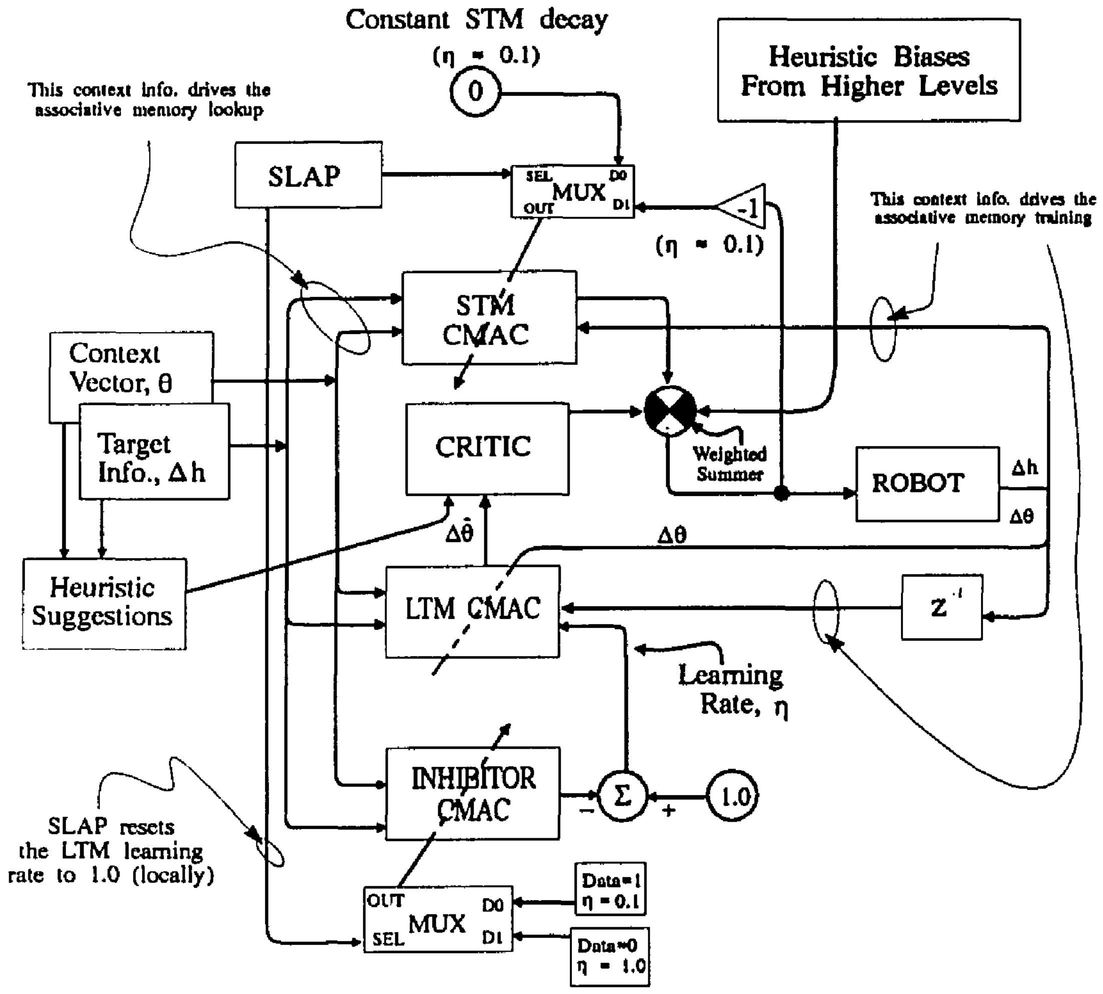


Figure 3.2: Overview of ARTFORMS-1

As a notational convenience, in the dataflow diagrams of this dissertation, the circuit symbol for a multiplexer (MUX) is used to replace a module like the truth-gate/false-gate dataflow construct of figure 3.1a. The purpose of a multiplexer is to allow alternate dataflows from one module to another that is controlled by some condition. For instance, in figure 3.2, the reinforcement signal, SLAP, controls the training into both the short term memory (STM) CMAC and the inhibitor CMAC. When the SLAP signal is dormant, the inhibitor CMAC receives a data signal of 1 as its exemplar value and is trained with a small learning rate[1] of $\eta = 1.0$. Whenever SLAP is active, exemplar data of 0.0 with a learning rate of 1.0 is gated through the MUX as an exemplar.

On the left-hand side of figure 3.2, input vectors flow in. These input vectors establish a context for the articulator that consists of a postural vector, $\underline{\theta}$, and a target vector, $\underline{\Delta h} = (\delta x, \delta y)$, consisting of direction cosines pointing from the hand to the target.

Direct inverse response vectors, $\underline{\Delta\theta}$, are then trained into the LTM CMAC near the center of figure 3.2. This CMAC's output will become $\underline{\Delta\hat{\theta}}$, the estimate of an inverse differential kinematic solution.

At the same time, a second CMAC, the INHIBITOR, learns the constant function, 1.0, with a small learning rate, as a function of the same input (context) vectors. This CMAC's output becomes a measure of the amount of experience the LTM CMAC has acquired as a function of the input vectors, and is used to compute a learning rate for the LTM CMAC. In this fashion, the INHIBITOR causes a gradual reduction of the plasticity of the long term memory (LTM) CMAC, by altering the latter's learning rate, or can quickly increase the learning rate of the LTM CMAC in response to an obstacle, in an effort to make the LTM more plastic. Without this plasticity control, the system's redundancy would allow it to drift from one valid trajectory solution to another. The act of settling on a particular trajectory in a particular context is a form of habituation.

1 Learning rate, $\eta$, is a measure of step size during the iterative convergence caused by execution of the delta rule. See section 4.6.4.5 on page 102 for discussion of the CMAC delta rule.

The method used to modify such acquired habits is provided by the reinforcement signal, SLAP, and its associated short term memory, the STM CMAC. In its *normal mode,* SLAP provides a zero vector as training exemplar for the STM CMAC, which enforces a gradual STM decay of any information that was already in the STM. The gradual nature of the decay is a result of a small learning rate for the STM training. If, however, a collision signal engages SLAP, the most recent inverse model supplied moves of the ROBOT are negated and trained into the STM CMAC with a large learning rate, of $\eta = 1.0$. The STM CMAC then emits a non-zero output to be summed into the forward activation supplied by the LTM CMAC as input to the ROBOT. The STM meanwhile decays to zero, via the normal mode of STM training just mentioned, so this perturbation triggered by SLAP will be transient. During the transient perturbation of the model, it is assumed that the system will be forced to learn some alternate solution for the inverse kinematics. Whenever a solution results in collision with an obstacle, it is perturbed from that solution, until some trajectory solution that is obstacle free becomes a habit in the given context.

Heuristics are used to generate suggested moves whenever the LTM either has no information trained into the memory associated with the current context, or if the LTM's suggested move violates goal directed conditions. The goal directed conditions impose a Lyapunov-like convergence condition on the system, by only allowing training with exemplars that reduce the hand to target distance, while discarding any non-convergent moves.

Critical to this system is the notion that a habituated trajectory will, in the redundant case, happenstantially often be appropriate to the problem being solved, and that the weighted summer that mixes the outputs of the various sources of trajectory information can be crafted in such a way that the selective disturbance of trajectories will also be appropriate. Both these assumptions have proved over-optimistic.

## 3.2 Back-track Search is Not an Explicit Part of ARTFORMS

The ARTISTS/ARTFORMS system is a shallow search method, having no explicit back-track capability. It uses hill-climbing or gradient descent as its principal goal direction mechanism. At any point, a heuristic or adaptive move suggestion for what to do next is required that will be critiqued by a *heuristic critic* or a *training critic*[2] based on whether or not the distance to the target has been reduced. If the proposed move will not reduce the distance, the suggestion is rejected. This criterion can be relaxed to some extent, but full blown back-track search is left to be addressed by a higher level in the planning architecture.

### 3.2.1 How Heuristics Enter The System

Heuristic search is thus exploited and then turned into a feedforward activation process by simply presenting steps postulated by a heuristic "suggestion" generator. As this process progresses, ARTISTS observes each move and its associated context vector, and trains the direct inverse LTM with that move as exemplar. This method, if viewed upstream of the heuristic critic, is clearly not goal directed; this objection has been raised by Jordan ['90]. If the heuristic move suggestions were merely randomly generated ones, serendipity would dictate whether any observed heuristic move would in fact be germane to the issue of decreasing the distance to the goal state. But if the heuristics used *are* goal di-

2 The two types of critics are described shortly.

rected, this objection loses strength. (See section 2.3.) After the critics' evaluations, goal directedness is stronger yet.

Any appropriate biasing influence, including obstacle avoidance suggestions, can easily be "piped in" from AI layers, impedance control modules, vision systems, minimum norm optimal methods, heuristic search methods like MURPHY, etc. Such sophisticated adjunct control modules are not necessary for merely adequate behavior. Each can simply subsume the heuristic suggestion generator whenever a higher level collision detector engages. In the event that the adequacy of this system's behavior is deemed insufficient, higher level help could be blended in to the level necessary to bring the behavior's adequacy up to some *desired* level. In this fashion a balance can be struck: only as much computational load as is minimally necessary need be added to achieve the desired level of competency.

### *3.2.2* ***Goal Directed Heuristics***

Heuristics that are applied before the plant is moved are *a priori* heuristics. Another set of *a posteriori* heuristics are applied after the fact by the heuristic and training critics. Two types of *a priori* heuristics have been successfully and extensively used: Random flailing and the Berkinblitt synergy.

*3.2.2.1* ***Random Flailing.*** Whenever the system has no previously learned knowledge at a particular state space location (i.e. when the LTM CMAC returns a near zero activation level) or if other methods have failed, $n$ random numbers within certain limits are cast. These become an $n$ dimensional joint change vector (for $n$ joints). This is not goal directed *per se,* but goal direction is *imposed* on the system by a critic as described in section 3.2.3, which prevents the system being distracted by erroneous random moves.

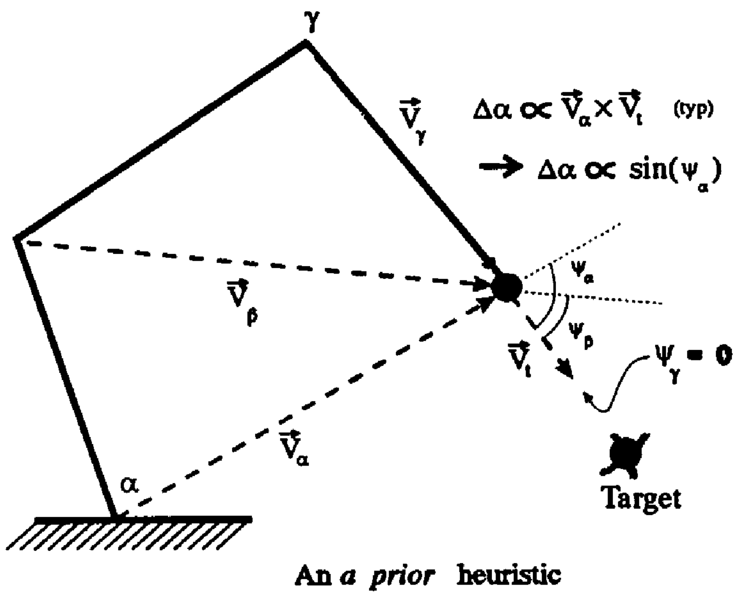


Figure 3.3: The Berkinblitt Synergy

*3.2.2.2* ***The Berkinblitt Synergy.*** The Berkinblitt synergy is based on observations of a spinal frog's wiping reflex. (See Handelman ['90] and Berkinblitt ['86].) It approximates straight line hand movement with approximately minimal total torque exertion, which would be important in a kinetics capable extension of this system. The algorithm is an approximate, qualitatively goal directed heuristic that computes very rapidly, and presents, open loop, a suggestion that reduces the hand-to-target distance.

*3.2.2.3* ***Description of the Berkinblitt Algorithm.*** The Berkinblitt synergy is described concisely in figure 3.3 At first glance it appears to be a solution to the inverse kinematics problem, but it is not. It is simply an interesting rule of thumb that gives a correct suggestion for any one-joint-only move, that will best reduce the hand-to-target distance, but when more than one joint is moved at a time, though the result will most always reduce the distance, it may not do so optimally. Consider, for instance, the example in the figure. By computing the cross products shown, it is clear that joints α and β should be adjusted by small negative angles. Angle γ should remain unchanged. Viewed statically and in isolation, each of these suggestions seem reasonable, but if α is decreased slightly, it is obvious that a slight simultaneous *increase* in β best reduces the hand-to-target distance.

*3.2.2.4* ***Other** a Priori **Heuristics.*** Other heuristics, besides random flailing, which are more goal directed, but still more efficient than search or constraint methods were tried, such as:

- Requiring a sufficient set of the joint angles' changes to be equal, so that the inverse problem becomes a non-redundant one.
- Exploiting synergies like opposite signs for selected pairs of neighboring joints (Hinton ['84]). This can, for instance, allow elbows to move while keeping the same hand position.

In fact these did not appear to be any more effective than random training, and the clear winner was the Berkinblitt algorithm. The policy that was finally implemented was to use Berkinblitt unless its suggestion was rejected by the critic, and then to revert temporarily to the random policy, which would always (eventually) succeed.

A third set of policies are discussed in sections 4.2 and 5.2.2 as the postural constraint and the central obstacle avoidance mechanism. The rules used to devise the objective functions for these constraint satisfactions are simply other forms of heuristics. Although these constraint rules are designed into the system *a priori*, their application occurs concurrently with training (and movement), thus their effect is neither *a priori* nor *a posteriori*.

If more heuristic policies were available, a flexible policy could be implemented that could intelligently select from among heuristic methods as need dictates. This policy could even be selected by a spatially distributed parameterization. (See sections 3.3.1.2, 4.8.1.1, A.1.1.1 and A.1.1.3).

*3.2.2.5* ***Relaxation of Goal Directedness: Hand Constraints.*** The heuristic goal direction is modified by changing the values of the hand constraint parameters $\varphi_0$ and $\varphi_{90}$ that were discussed in section 2.4.3. The adjustment of these angles can allow the search for trajectories to proceed using heuristic moves that vary from the rectilinear hand move constraint by any arbitrary amount. A very interesting qualitative result was observed: for non-redundant arms (i.e. 2 joints) any values of $\varphi_{90}$, worked. For large values of $\varphi_{90}$, the resultant search process was quite curious to watch. In those cases, the hand at first followed bizarre looping trajectories and seemed unlikely to settle on reasonable trajectories. After a single pass around the targets, however, the arm settled down into trajectories that were nearly rectilinear and improved rapidly. The amount of saturation of memory went up during the peculiar looping exercises in such cases. (See the discussion of memory usage in section 4.8.10.)

### *3.2.3* ***The Heuristic Critic***

The heuristic critic predicts (as described in section 2.4.3) whether or not a heuristically or randomly derived move suggestion will result in a desired degree of goal directedness. An analytical model of the forward plant provides a perfect prediction each time, but this need not be so. (See section 4.8.12 on page 132 and section 6.3 on page 159).

*3.2.3.1* ***Assumption of a Nearly Reversible Plant.*** The use of the plant as model for the critic is perfectly acceptable if there is a reversible plant.

If the plant is not exactly reversible, it will not be a problem unless reversal of a move causes goal divergence that sweeps out completely virgin state space (i.e., state space that is previously unvisited directly or through generalization). Indeed, it may be problematic for a physically realized ARTFORMS system if the physical plant were not at least *nearly* reversible for small incremental moves.

On the other hand, it may be quite acceptable to just allow the plant to make goal divergent moves without bothering to reverse them. It suffices to say that the current ARTFORMS-1 and ARTFORMS-2 are intended to be idealized limiting cases for such planning systems, hence the use of the analytical forward model in the critics. The development of more realistic systems is left as future work, with the discussions of section 4.8.12 on page 132 and section 6.3 on page 159 finishing the current discussion.

### *3.2.4 The Training Critic*

The training critic is activated to critique every move that has been read from the LTM to determine if it is appropriate. The same world as model paradigm is used for the training critic as was used for the heuristic critic. It was observed in all successful simulations that only during the first few segments did the training critic fail, so its absence is not a problem after early training. It *is* needed in two situations:

- During early training, before the LTM converges.
- During very late training if memory has been so saturated that hashing damage occurs. Such occurs when the CMAC memory is sized too small. For discussion of when the hashing damage is transient and relative innocuous, see section 4.8.10. For a discussion of experimental results using an inaccurate

model for both the heuristic and training critics, see sections 4.8.12, and 6.3.

#### *3.2.4.1* ***Step Size Control.***

Associated with the training critic, there is a need to ensure that hand moves are uniform length, as a cue for when to reject low grade data whose genesis might be an artifact of hashing collisions. To this end, and also to the end of providing a more uniform addressing of state space, any move that is postulated by the CMAC or the heuristic move generator is tested on the plant to see if it in fact generates a unit length hand move. If the move is outside certain limits around a nearly unit length then an iterative process of scaling the joint move until these limits are satisfied is executed. This is a flagrant appeal to the linearity of inverse differential kinematics in which Appendix C gives us some faith. It was determined experimentally that 0.8 to 1.2 were acceptable limits for approximate unit length of steps. Tighter limits caused too much time to be wasted in iteration. Looser limits may have been acceptable, but these limits worked well in practice.

## *3.3* ***Multiple CMACs: Spatially Distributed Parameterizations***

The concept of a spatially distributed parameterization is a crucial concept of this dissertation. The idea will be visited over and over again. The rest of this chapter describes the first generation attempt at an implementation of obstacle avoidance using this means. In section 4.4 the problems that led to rejection of the first generation solution are described. Some readers may wish to skip immediately to chapter 4 and proceed to the ultimate second generation system. To preserve the chronology of this development, the first generation solution is left in. It still contains many solid concepts, especially relative to spatially distributed parameterizations. The first such parameterization we shall discuss is a spatially distributed plasticity mechanism.

### *3.3.1* ***Habituation***

This supervised learning system incorporates three CMACs, trained concurrently using the same inputs. The first one learns the inverse model. Another one, the *inhibitor*, is used to stabilize the first such that it robustly learns sub-optimal trajectories through hand space. The third CMAC network is involved in obstacle avoidance.

A successful trajectory goes from start point to target point without getting stuck in between, and without violating some imposed conditions. Assume that whenever a successful trajectory is observed it is "recorded" in the inverse model network. Suppose the trajectory can be reliably "replayed" by simply starting at a point on that trajectory and asking the inverse model network to recall the sequence of moves of that trajectory. If this can occur for any arbitrary successful trajectory, then the system clearly can learn sub-optimal trajectories.

*3.3.1.1* ***The Inhibitor Network.*** A good definition of habituation is the tendency to execute a particular action in a given context only because that same action or one similar to it has been done before in the same or similar context.

Habituation is strengthened over time by the activity of a second, "inhibitor" network, which stabilizes the inverse model network. It schedules the inverse model's learning rate down in high usage regions of state space while leaving the rate large (i.e. 0.5 or so) elsewhere. This may not allow final convergence of a *particular* sub-optimal trajectory, which is a desirable and exploitable feature of this method, because an *adequate* trajectory with an obstacle present may be *far* from the optimal trajectory aimed at the same target but without the obstacle. The mechanism is simple: the inhibitor is presented with the same input vector as the inverse model. It is trained with the scalar response function $r(\theta)=1.0$. The inhibitor network's learning rate, $\eta_{inh}$ is small. This allows the inverse model network time to develop trajectories before the trajectory is fully habituated or "frozen". The larger $\eta_{inh}$ is, the faster trajectories habituate. Obviously selecting a value of $\eta_{inh}$ is critical. The inhibitor will then output $0 \leq \hat{r}(\theta) \leq 1.0$, which is used to construct the learning rate, $\eta = 1 - \hat{r}(\theta)$, for the next training cycle of the inverse model.

*3.3.1.2* ***Structurally Equivalent CMACs.*** The inhibitor network operates concurrently with the inverse model network and is structurally identical to the inverse model network, in that it has the same input vector, $\theta$, as the inverse model (i.e. it then has the same number of degrees of freedom). It has the same "internal wiring", by virtue of having the same generalization, hashing algorithm, and address decoding algorithm. So, instead of actually allocating a second CMAC, a more efficient implementation *may* entail including the inhibi-

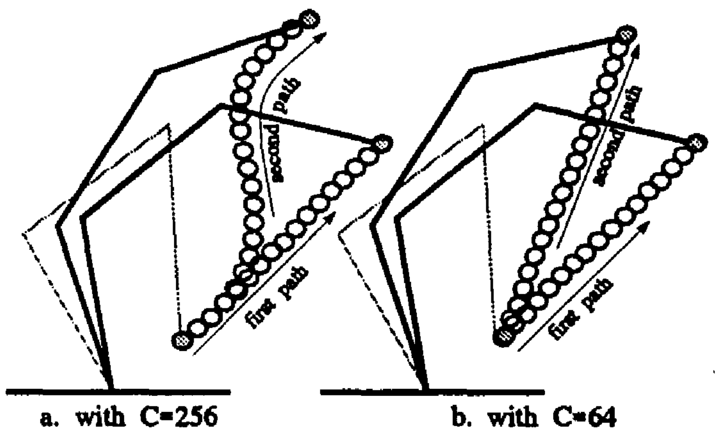


Figure 3.4: Effect of Generalization on Trajectories

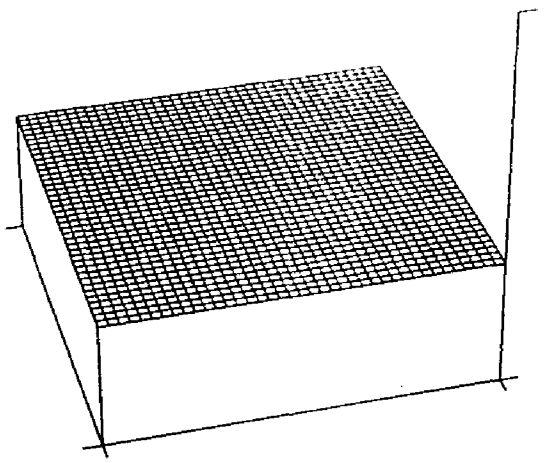
tion level as a fourth component of the response vector trained into the inverse model network.

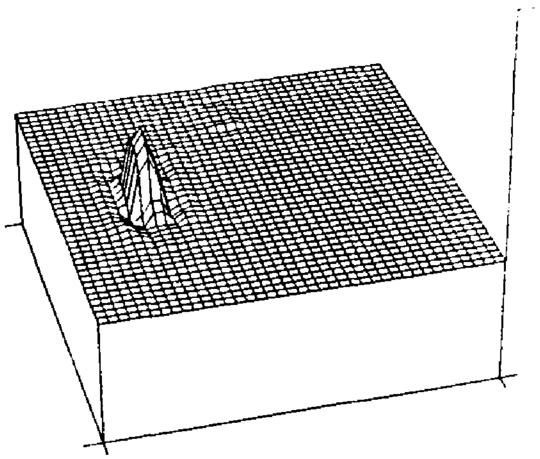
It may, however, be argued that the inhibitor network should have less generalization than the inverse model, requiring a separate CMAC for its implementation. See figures 3.4a and b. For the case of large generalization, the formation of the first trajectory strongly influences the formation of the second. In instances where two nearby trajectories *should* be pulled apart due to an interstitial obstacle we would want the first trajectory to *influence* but *not dictate* that nearby trajectories emulate its form. In other words, by making $C_{inh}<C_{ltm}$ we are attempting to prevent regions that have been visited *only* through generalization and not *direct* exemplar training to remain plastic longer. This notion of a *spatially distributed plasticity measure* is a central theme of this thesis and is one of its major original contributions. In fact, a CMAC is an ideal tool for a field representation for any spatially distributed parameterization. This method is a significant means for increasing the power and flexibility of parameterization for adaptive systems in general. To date, the only similar application is Moody's ['89] method of cooperative interconnection of multiple resolution CMACs. Moody's method was put forth only as a learning speedup mechanism.

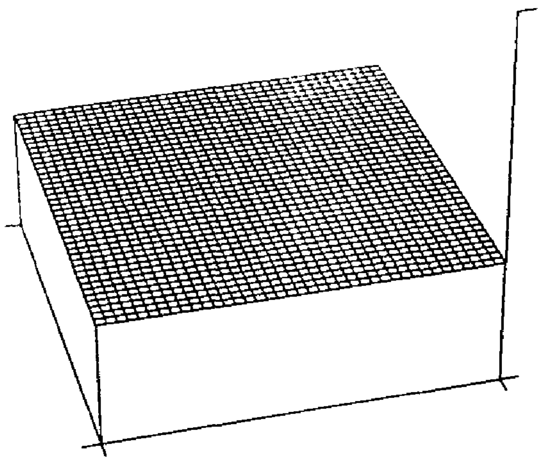
## *3.4* *Reinforcement Learning: Modifying Habits*

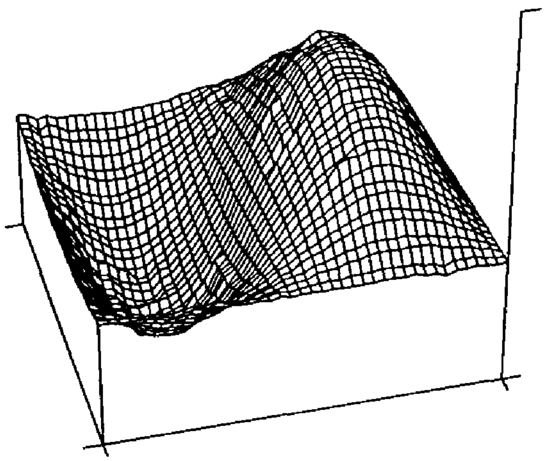
The ARTISTS architecture becomes ARTFORMS with the addition of reinforcement learning. A third, "repeller", network is subsumptively connected to the other two. In the presence of a reinforcement signal (SLAP) it is trained on exemplars that are the negations of the most recent inverse model moves, and its learning rate is large (near 1.0). It locally inhibits the inhibitor network (i.e. trains it to zero, with a learning rate of 1.0), while summing a negated move into the inverse model's training example. This action, taken repetitively, overcomes habituation in a local area and causes the system to relearn new trajectories when obstacles are encountered. Succinctly this network learns to "do the opposite" of whatever was recently done that tended to get the system into trouble, as indicated by the reinforcement signal. Since the inhibitor network has been trained down to a near zero inhibition level in this neighborhood, and since the output is summed with the training signal for the inverse model, then on subsequent training passes, the reverse action emitted by the repeller is propagated *backward and forward* along the trajectory to the extent that generalization allows. This repeller CMAC can be viewed as a short term memory module, (STM). In addition to the training described above, it is being trained at every visited state space point with a zero vector as training exemplar and a small learning rate in order to effect the memory decay necessary for an STM. The direct inverse model is a long term memory, (LTM).

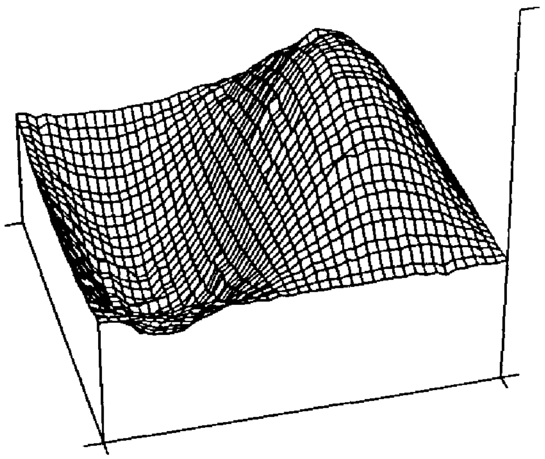
This "habituation" paradigm stops and starts convergence toward sub-optimal trajectories, to allow the "elbows" to migrate around, with or without hand disturbance, to avoid obstacles as the repeller STM's information is transferred to the LTM. This allows the joint postures to naturally assume positions related to recently visited postures via generalization. The final result is that obstacle avoidance maneuvers eventually become part of an overall direct inverse model of the inverse kinematics, consisting of three networks operating in parallel. Figure 3.2 shows this model. It is instructive, however, to consider a simpler analog of the model, as follows.


(a) STM memory before reinforcement training has been triggered.


(b) STM memory after a reinforcement trained feature is trained in


(c) STM memory after the reinforcement training feature has decayed away


(d) LTM memory after initial training


(e) LTM memory during reinforcemen training - no change yet.

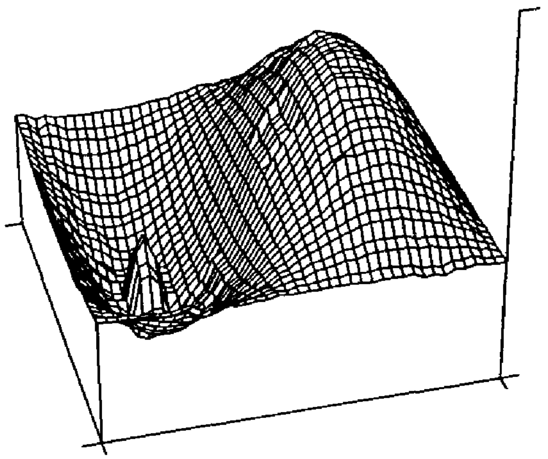

(f) LTM memory after the feature or "bump" has projected from STM to LTM.

Figure 3.5: STM to LTM projection

### 3.4.1 A Simplified "Introductory" System

To more clearly explain the ARTFORMS system, a system is presented that is called 2DTFORMS whose postural dimensionality is 2, so that the LTM can be easily visualized using plots like those in figure 3.5. Unfortunately this system is not posturally redundant, so it's input problem will not be a perfect idealization of the task that ARTFORMS must perform, but it will suffice to illustrate the basic concept involved in the STM $\rightarrow$ LTM projection. Understanding of this should lead to a complete understanding of the ARTISTS + ARTFORMS system as summarized in figure 3.10 for 2DTFORMS, and ultimately in figure 3.2 of page 55 for ARTFORMS-1.

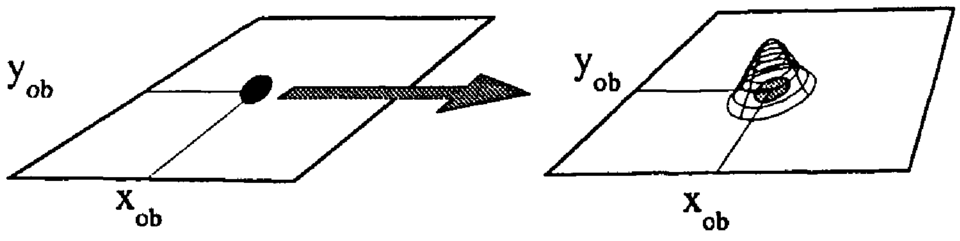


Figure 3.6: Formation of a Feature in STM

### 3.4.2 A Short Term Memory Obstacle Representation

Consider figure 3.6. This illustrates a convex perturbation or *bump* function being formed in the STM state space around a perceived obstacle in 2-space. Now consider figure 3.5: the first two panels on the bottom are approximations of the function $z(x,y)=sin(x)sin(y)$. The figure is actually the output, $\hat{z}(x,y)$ of a CMAC that has been trained on a limited range of $(x,y)$ inputs. By training this direct inverse model as shown in figure 3.8, $\hat{z}(x,y)$ becomes a very close approximation of $z(x,y)$, and can serve as an LTM of a trajectory model. The trajectories that it can produce are like those in figure 3.9, which are phase portraits of the time response of the direct inverse system shown in figure 3.7. The forward model is just the pair of difference equations shown in figure 3.7. The bell shaped trajectories produced by this simple system take the state point from a negative value of x to a target point which is any point on the right margin of the x-y plane shown in figure 3.9. This should not be confused with any robotic manipulator; it is purely a mathematical exercise. Suppose for some reason we wish to disturb the regular set of trajectories emitted by this model. All we desire is that if the state point moves along the trajectory T as shown in figure 3.9, before the obstacle at O is encountered, the system should change modes so as to jump to a "neighboring" bell shaped trajec-

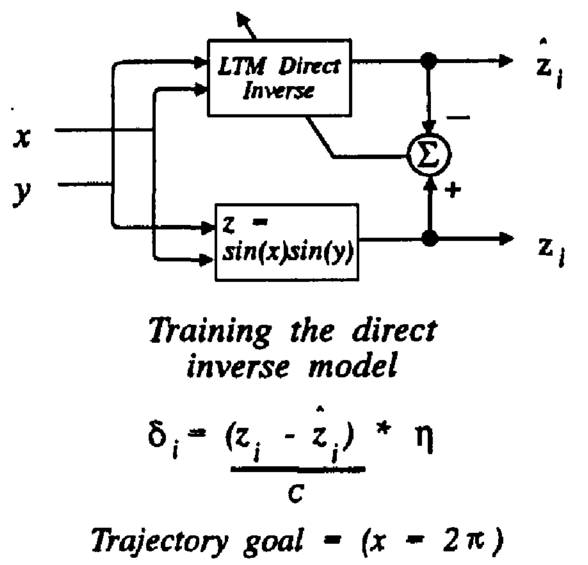

LTM Direct Inverse
z = sin(x)sin(y)
x
y
ẑ_i
z_i
Σ
−
+
Training the direct inverse model
δ_i = (z_i − ẑ_i) / c * η
Trajectory goal = (x = 2π)

Figure 3.8: Training a Direct Inverse Model

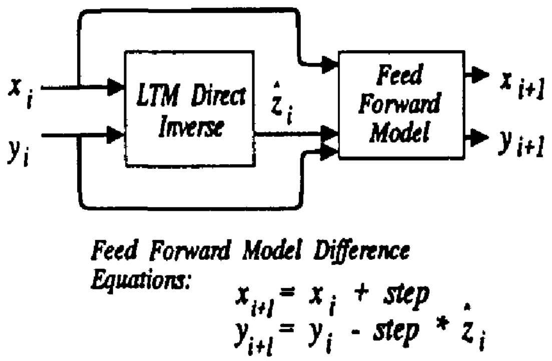


Figure 3.7: A 2D Non-Redundant Trajectory Planner.

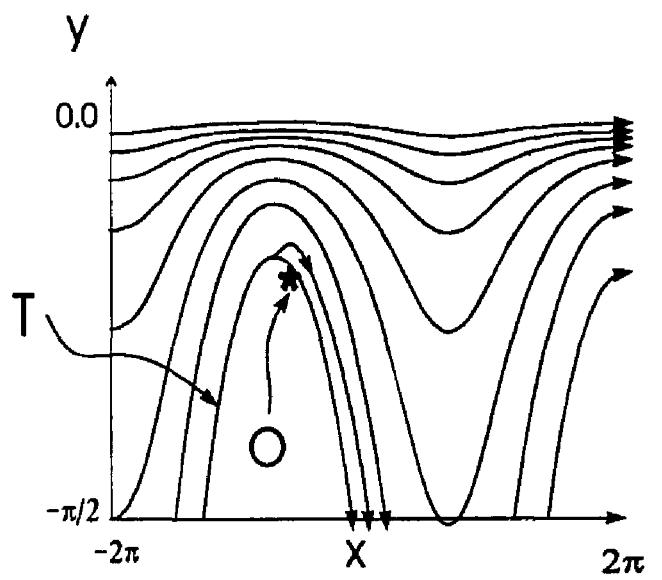


Figure 3.9: Phase Portraits of 2DTFORMS

tory. So the system must learn a bump function that will perturb the trajectory from T to a neighboring trajectory. The system must then stably remember to do the same thing whenever it encounters a point near that obstacle, so that T and very nearby trajectories will all jump up to "higher" (in the phase portrait) trajectories to avoid obstacle O whenever encountered.

The difference equations of figure 3.7 are equivalent to:

$$J_{xy} = \begin{vmatrix} 1 & 0 \\ 0 & -sin(x) \cdot sin(y) \end{vmatrix} \text{ and}$$

$$S_{i+1} = S_i + \Delta t \cdot J_{xy} \qquad (3.2)$$

where $\Delta t$ is the step size along the trajectory, $J_{xy}$ is the Jacobian of the trajectory ensemble and $S_i$ is the $i^{th}$ point, *(x,y)* along the trajectory. Clearly there are many paths from $(x_0,y_0)$ to the target point $(2\pi,y_T)$. Equation (3.2) defines adequate *(but certainly not unique)* trajectories meeting this criterion.

### *3.4.3 Training the LTM*

We may start training our system with a null LTM. At every step, we train as we step along trajectories. In early training, performance may be erratic. In fact we may choose to only "think about" executing trajectories at first. In the event this *were* a planner for a physical actuator, this would avoid damaging a device that may be intolerant of or unable to realize the trajectories so formed. During this time we would then be relying on the "world as model" by actually computing the forward model each time, since it is easily computable. We should eventually arrive at a stable trajectory set, because $sin(x)sin(y)$ is deterministic and single valued. At some point along the way we should be able to rely on the output of the LTM. Training so far has proceeded as in figure 3.8.

### *3.4.4 Training the STM*

Suppose that we have a second CMAC that is to model an STM. It is initially null. We always train the STM on the function $f(x,y)=0$, with a slow learning rate of about 0.1 (for instance). This would represent a constant STM decay. Let us assume that the direct inverse training has $\eta=0.5$. If during trajectory formation an obstacle is encountered, the STM is trained using as exemplar the function, $f(x,y) = k$, for constant $k$, with a learning rate of $\eta=1.0$, to cause rapid formation of the STM "bump" or perturbation function as shown in figure 3.6. The actual feature trained into the CMAC would not be as smooth as the bump shown, but it should suffice as an approximation of a convex perturbation function. The important point is that it does not affect trajectory states anywhere more than $r \approx C$ codon units away from the obstacle, $(x_{ob}, y_{ob})$, where C is the generalization parameter of the STM CMAC. Codon unit is defined in section 2.6.2.2.

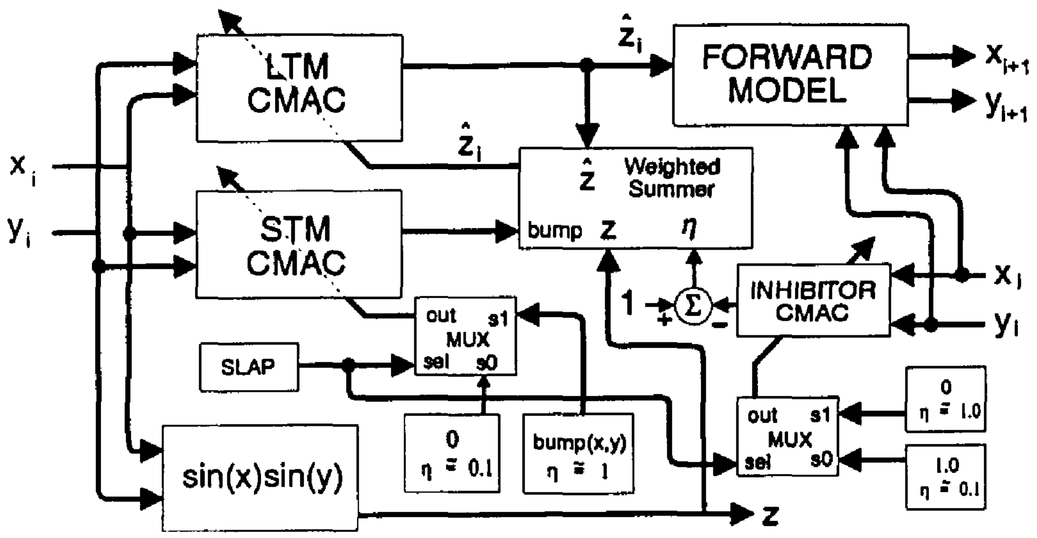


Figure 3.10: A 2D Simplified Analog of ARTFORMS-1

### *3.4.5* ***Projecting the STM onto the LTM***

Rather than referencing the LTM, we reference LTM(*x,y*)+STM(*x,y*). At the same time, we train the LTM based on the observed moves along the trajectory. In the process of this training, what happens is that the bump function is transferred into the LTM in a manner described by the time sequence in figure 3.5. It is critical to adjust the learning rate during training in such a way that the bump function transfers from the STM to the LTM before the STM decays to zero and that the STM decays to zero soon enough that its presence in the summed value that becomes the training signal for the LTM does not unnecessarily amplify the magnitude of the perturbation. It is clear that the weighted summer box of figure 3.10 is more complex than the figure indicates! This box is responsible for using *only* the *sin(x)sin(y)* function as an exemplar when the LTM is untrained. The weighting for the summer is controlled by having a context sensitive variable learning rate, η(*x,y*). This context sensitivity is provided by *another* CMAC. This CMAC is the inhibitor CMAC. It is trained in parallel with the LTM (using the same input state vector) and is trained with 1.0 as exemplar function. So if INH($x$,$y$) = 0 it indicates that $\eta$ for the LTM should be large *at that point.* If on the other hand INH($x$,$y$) = 1.0 it indicates that $\eta$ for the LTM should $\rightarrow 0$, which establishes a condition of absolute trajectory stability for that context, and other values of INH($x$,$y$) indicate intermediate values of $\eta_{inh}$.

### 3.4.6 Habituation Can Be Disturbed

When a condition of absolute trajectory stability is established at a point, ($x$,$y$), there are still 2 mechanisms whereby the value of LTM(x,y) could be perturbed:

First, consider the effect of overlapping locally receptive fields along a trajectory: if a state vector ($x+\delta$, $y+\varepsilon$) occurs during training, where $\delta+\varepsilon < C_{ltm}$, and $C_{ltm}$ is the LTM generalization, then training at LTM($x+\delta$, $y+\varepsilon$) could affect LTM($x$,$y$).

Secondly, during training at *any* point, it is possible, as discussed in section 4.3, that hashing collisions could cause spurious generalization that can, over time, affect LTM($x$,$y$). The former cause is not a problem, in fact it simply reinforces an innate smoothness constraint that CMAC provides the model through generalization. The latter cause is problematic though in that over time a trajectory may drift due to hashing, as more and more trajectories are learned. This means that constant vigilance is necessary. The sensors that feed the SLAP signal can never go to sleep. As long as the physical memory is large enough, though, the occurrence of SLAP should, on average, decrease over time. If this last condition is not met, the remedy is to enlarge physical memory of the LTM.

### 3.4.7 *Discussion of 2DTFORMS*

Taken as a whole, the system just described causes trajectories that pass near the obstacle, O, in figure 3.9 to be deflected as shown in that figure. This overall behavior is very similar to the desired behavior of ARTFORMS.

By now other properties of the system become apparent:

- The magnitude of $C_{stm}$ determines by how much trajectories will be deflected from nominal when SLAP occurs. The same reasoning applies here as in section 3.3.1.2 regarding how old trajectories can distort new ones through generalization.
- There is an innate smoothness constraint imposed on a system reliant on a locally generalizing memory, because generalization will propagate postural suggestions forward and backward in time along the trajectory. This causes the following: after trajectory T has been learned and obstacle O is first encountered, during subsequent practice of T, the deflection will occur *before* the obstacle, and thus before SLAP occurs.
- If we think of the extrapolation of this system to a controller for a physical manipulator, the resolution of the STM can probably be much coarser than for the LTM, because the effect of obstacles should on average be less fine grained than the desired precision of the mechanism. The reason for this is that any mechanism probably has members that are *thicker* than the smallest move the end point is intended to make. Thus, if an obstacle affects it similarly for a given posture, it will affect it for postures near that point by a distance equal to at least the thickness of the mechanical links.

If an obstacle occurs at ($x_{ob}$, $y_{ob}$) the STM quickly learns to avoid it by virtue of training in a convex function centered about ($x_{ob}$, $y_{ob}$) in the STM model. The radius of the function should be $r \approx C/2$ so it can be learned virtually completely in a single training instance with learning rate of 1.0. As the bump is trained into the LTM by reinforcement learning, that region of the STM is allowed to decay back to null by retraining it with the exemplar function 0.0. During this reinforcement learning, the LTM and STM are summed to form exemplars for retraining the LTM. In this fashion, the LTM retains the superimposed image of the bump while the bump disappears from the STM.

The analogy between the simple 2D STM $\rightarrow$ LTM projection model (2DTFORMS) and ARTFORMS breaks down here. Since ARTFORMS is posturally redundant, and not just path redundant, it has multiple stable trajectories between any 2 state points that can be formed and used as training exemplar sources, directly from the plant, whereas 2DTFORMS does not. By having to artificially rely on the analytical *sin(x)sin(y)* model to train the LTM, the analytical model can not be used to train the LTM once a "bump" has been superimposed on the LTM image of the *sin(x)sin(y)* model, because if such were the case, the bump would decay from the LTM just as it does from the STM.

## *3.5 Heuristics and Habits: An Action Compiler*

The weighted summer for 2DTFORMS must cause the training examples for the LTM to be the ideal model *sin(x)sin(y)* whenever the LTM is immature and SLAP is not present, and to be the LTM($x$,$y$) + bump($x$,$y$) if the LTM is mature or SLAP is present. In the case of the robot arm, there is no ideal function to refer to. Rather this role is fulfilled by any heuristics that are presented, *deux* *ex machina* fashion from a higher level. The heuristic that is applied may be an analytical solution (which is heuristic because it must, by definition, have a heuristic constraint satisfaction applied to get an inverse solution of the redundant problem), or it may be supplied by an AI system that has common-sense rules embedded in a knowledge base or blackboard, or it could be random "flailing" around suggestions, in the event that nothing more interesting can be contrived in the current instance. This is a form of *action compilation* whereby difficult symbolic or analytic computations used to arrive at desirable activities can be converted into reflexive actions. Handelman, *et al* ['89], investigated a related method. We exploit certain properties of CMAC modules here, forming a larger scheme that mimics what is actually done in a cerebellum. Simply stated, another central idea of this thesis is: *the conversion of heuristics into habits is an effective planning and learning paradigm.* To that extent, this system is seen to be based on a biological metaphor.

Suppose that ARTISTS has successfully habituated on a trajectory, T1. As ARTFORMS tries to deform trajectory T1 into T2, the system naturally habituates on T2 by sculpting "hyper-bumps" superimposed on the trajectory forming hypersurface in the LTM to cause trajectories to conform to the suggestions passed down from higher levels, including the reinforcement signal. But there is more going on here -- The system under control imposes its own innate ascending constraints on the system in that the dialog ARTFORMS observes is what the robot *actually does* in the current context, not what was requested, so SERVO level response limitations and physical postural limitations are automatically learned by the system. (See section 4.4.1.2.) That is to say T2 gets transformed into T2', which may have features in it that do not conform to the robot's hierarchy. This characteristic of the system is viewed as subsumptive in nature, in that ascending, lateral and descending flows of information from largely autonomous sources merge together in a fashion such that one or more sources can subsume the data flow path into the control surface.

## *3.6 Some Conclusions About The First Generation Solution*

In practice, ARTISTS was successful. Detailed experimental results are in the next chapter. There were problems associated with development of consistent trajectories and problems getting a reasonable implementation of ARTFORMS (ARTISTS + obstacle avoidance) to work at all. These deficiencies and a better 2nd generation architecture, called ARTFORMS-2 are discussed in the next chapter.

It is important to remember that what the robot *does* must be similar to what was requested for generalization to allow any meaningful learning to take place. This is a restatement of the need for goal directedness to be present in the training exemplars.

*Chapter IV*

# *The Second Generation Solution*

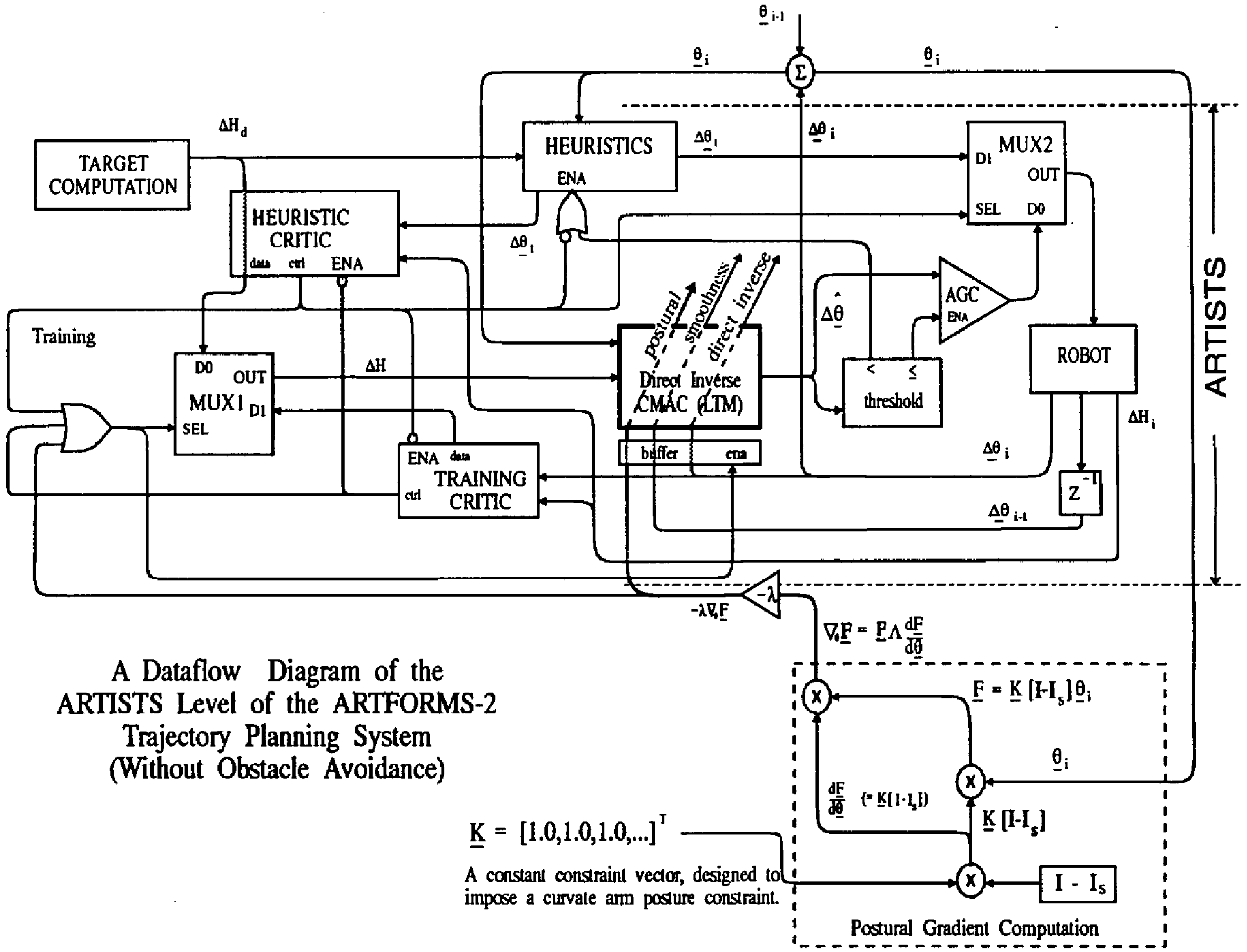
The failure of ARTFORMS-1 to live up to expectations led to the development of ARTFORMS-2. The sources of deficiencies and the necessary implementation of postural constraint satisfaction are developed in this chapter.

## *4.1 Preview: A Concise Description of the Architecture*

This chapter introduces a new planning method based on some of the principles introduced in chapter 3. In this chapter, however, the postural ambiguity is dealt with in a structured way so that trajectory formation becomes consistent and repeatable.

Figure 4.1 consists of an upper region labeled ARTISTS and a lower region, that consists of a gradient descent computation which is used, as described later in this chapter, to reduce the degrees of freedom and allow consistent inverse kinematic solutions for generating trajectories. The upper part is very closely related to the ARTISTS level described in chapter 3. everything above the dotted line in the figure is essentially preserved from the first generation solution.

Figure 3.2 on page 55 is similar to figure 4.1, with the principal exception that the latter is more detailed. A similar diagram appears in chapter 5, and there it will include a module that provides obstacle avoidance. In addition to the MUX elements used here, as in figure 3.2, there are also OR gates used where multiple control signals may affect the behavior of other elements.



A Dataflow Diagram of the ARTISTS Level of the ARTFORMS-2 Trajectory Planning System (Without Obstacle Avoidance)
TARGET COMPUTATION
HEURISTICS
ENA
HEURISTIC CRITIC
data ctrl ENA
MUX1
D0 OUT D1 SEL
MUX2
D1 OUT SEL D0
Training
TRAINING CRITIC
ENA data ctrl
Direct Inverse CMAC (LTM)
postural
smoothness
direct inverse
buffer
ena
AGC
ENA
threshold
< ≤
ROBOT
ARTISTS
$\Delta H_d$
$\Delta H$
$\Delta H_i$
$\underline{\theta}_{i-1}$
$\underline{\theta}_i$
$\Delta\underline{\theta}_i$
$\Delta\underline{\theta}_{i-1}$
$\Delta\hat{\underline{\theta}}$
$z^{-1}$
$\Sigma$
$-\lambda$
$-\lambda\nabla_\theta\underline{F}$
$\nabla_\theta\underline{F} = \underline{F}\Lambda\frac{d\underline{F}}{d\underline{\theta}}$
$\underline{F} = \underline{K}\,[I-I_s]\,\underline{\theta}_i$
$\frac{d\underline{F}}{d\underline{\theta}}$ $(=\underline{K}[I-I_s])$
$\underline{K}\,[I-I_s]$
$I - I_s$
X
Postural Gradient Computation
$\underline{K} = [1.0,1.0,1.0,...]^T$
A constant constraint vector, designed to impose a curvate arm posture constraint.

Figure 4.1: Dataflow of Second Generation Solution

Starting in the upper left corner of the figure, note that a target computation generates a desired hand move (direction vector) which becomes part of the input stream to MUX1 and the heuristics generator.

The training and heuristic critic modules mutually inhibit one another. This should only be interpreted to mean that one or the other module may evaluate a move. Both modules would never evaluate the same move. This is because the heuristic critic evaluates moves suggested by the heuristics generator, and the training critic evaluates moves suggested by the LTM CMAC. The output control lines from these two critics control the training signal gate and the heuristics generator. The control outputs are positive logic success or failure indicators.

Success for the critic means a suggested move met the current heuristic criteria. (See section 2.4.3., on page 33.) If the heuristic critic fails, then the heuristics generator is activated, causing a new move suggestion to be generated. If the heuristic critic succeeds, it enables the training signal gate, allowing the next move of the robot (which will be the result of its trying the current heuristically generated move suggestion) to become a training exemplar (provided that the training critic doesn't block its use as an exemplar).

The training critic receives data directly from the robot or a model of the forward mechanics of the robot, and evaluates whether a real or imagined move met the current criteria for success. If so, that move becomes part of the input data for the next training cycle and the training gate is enabled.

If the LTM CMAC receives input, it will be activated to output its current contents of the memory vectors associated with the current context; this phase is called lookup. If the LTM CMAC is activated and at the same time, the training gate is enabled, the training signals are gated in and a training cycle ensues; in other words, the CMAC is not required to output a signal, only to update the memory vectors associated with the current context using the current exemplar observed from the robot's behavior.

The input to the robot (or robot model) is multiplexed by MUX2. Thus its input can come either from a heuristically generated move, or from the LTM CMAC. If the data comes from the LTM CMAC, a stepsize control critic evaluates the Manhattan length of the step. If it is above a threshold size, indicating that it is true data, and not just a weak collateral generalization effect or hashing collision generated data, the AGC module is enabled and the step is scaled up to represent a near unit length in the hand space. If the heuristic critic is inactive, this data will become the next step. If the threshold logic failed, the AGC module is blocked and a new heuristic step is requested by activating the heuristics generator.

The lower right part of the diagram performs a postural gradient computation, which supplies a training step that is based on the current context ($\underline{\theta}_i$) and a constant curvate arm constraint vector, $\underline{K}$. This module will, on every training step, insert a training cycle that attempts to adjust the value of $\underline{\Delta\hat{\theta}}_i$ formed from the contents of the currently selected LTM memory vectors, to cause $\underline{\theta}_i$ to meet the curvate constraint.

## 4.2 ***Weaknesses of the First Generation Implementation***

The three weaknesses of ARTFORMS-1 are the stable trajectory problem, postural drift and a complete lack of goal directedness in the obstacle avoidance method. All these problems are discussed separately below.

## 4.3 ***Stable Trajectory Problem: Trajectory Drift***

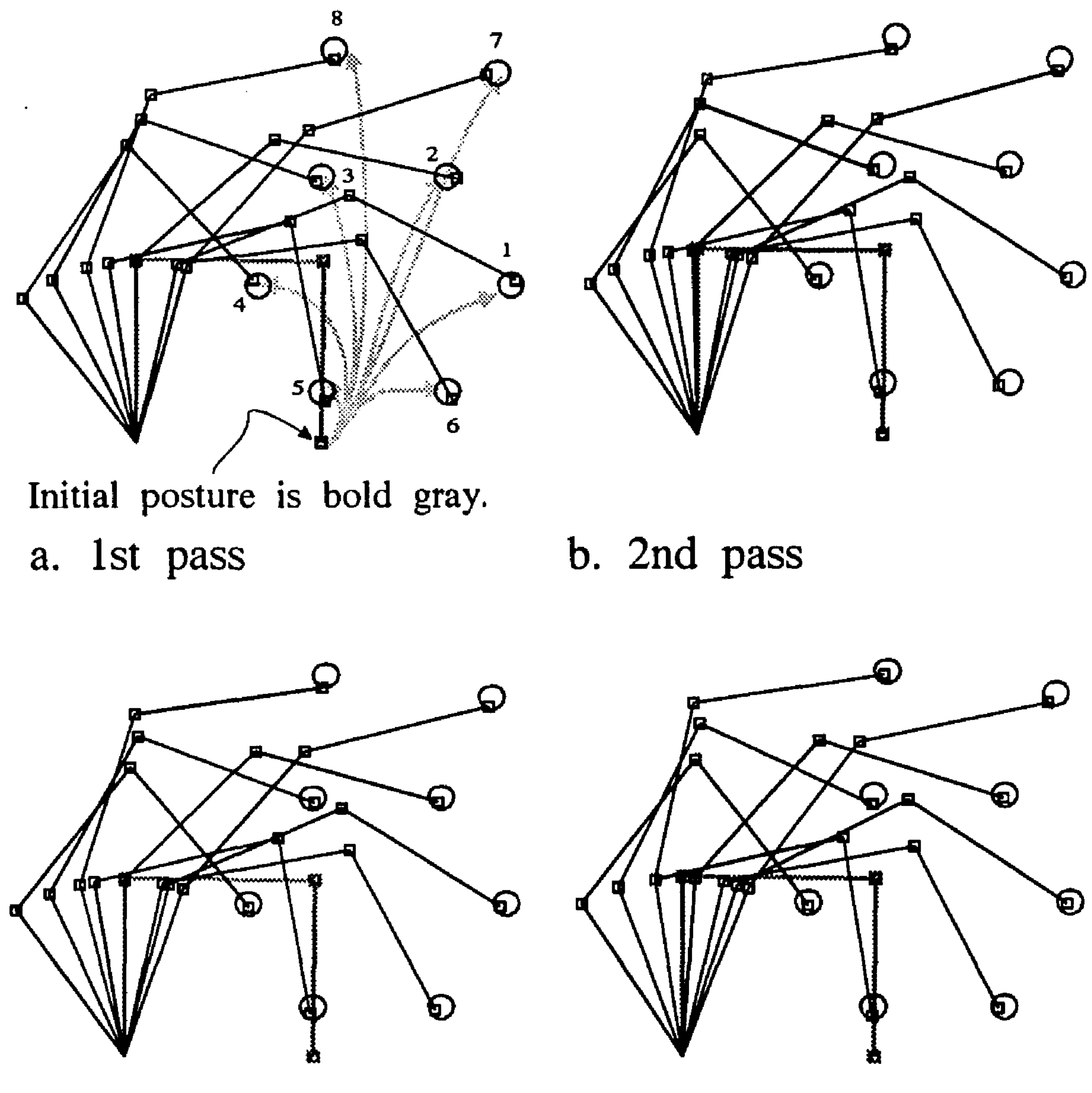
In both CMAC and MLP architectures there is some degree of distant point generalization that makes on-line learning problematic. Suppose an MLP uses a particular weight, $w$, in training for a posture $p$. The same weight, $w$, may then be enlisted during training for a geometrically distant posture, $p'$. This will require regular reinforcement of $p$, even when it is not an application relevant training example, just to prevent new training from disturbing old training. This becomes exponentially problematic as on-line training progresses. Weight competition is exacerbated for reasons related to global generalization by the need to scale down an MLP to a minimal size for computational speed. Though there *is* local generalization in MLPs as well as global (see section 2.6.2), there is competition for weights in response to training exemplars that is inherent in having to model large input spaces in small networks. This network scaling problem is central. When one considers that typically MLPs model dozens of nodes with perhaps hundreds of weights, the problem of selecting a network small enough to be practical, but large enough to prevent the weight competition just mentioned clearly reduces the feasibility of on-line training for MLPs.

A CMAC can also exhibit this kind of competition for weights, as a result of the hashing randomization that allows hundreds of thousands or even millions of "virtual" weights to be modeled in tens of thousands of physical weights, but it is a very low grade effect and only becomes important when the CMAC memory size is scaled too small for the problem. However, over time, this may cause fully habituated trajectories to drift. Having fully habituated, ARTFORMS-1 will be unable to correct this drift through further training. This sort of drift is unavoidable, and detection and correction of it poses a computational problem for ARTFORMS-1.

## *4.4* ***Postural Drift***

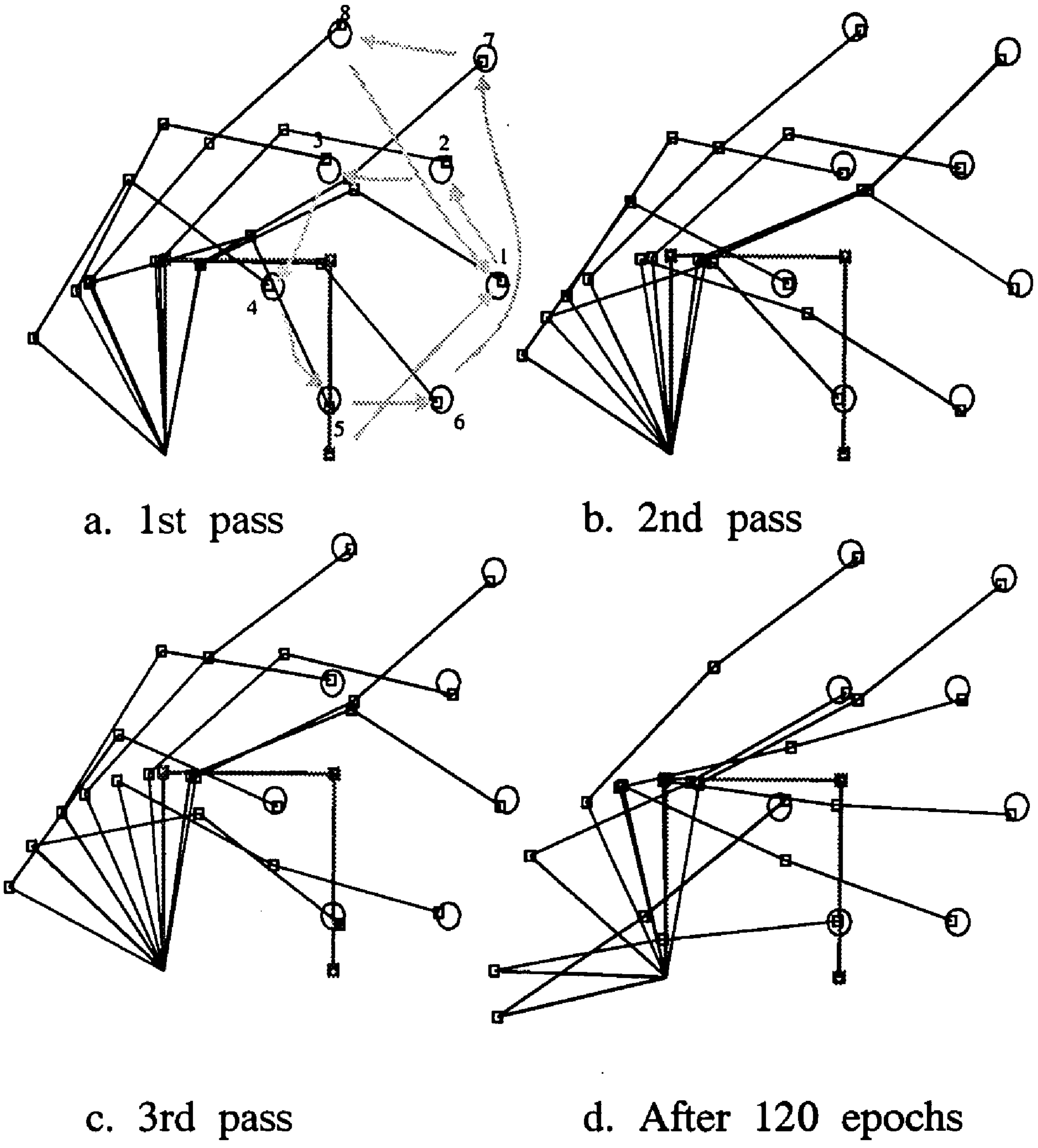
The central failing of ARTFORMS-1 was that it exhibited postural drift. When I first implemented ARTISTS, I always designed trajectories that started from a common initial posture and sequentially reached from that posture to each of the targets in turn. This method produced uniformly good results. In these cases, I also routinely started the simulations from a normal curvate posture. A typical result looked like the animation panels of figure 4.2. At that point I thought ARTISTS had achieved stable trajectory formation. Then I installed trajectory chaining, and the trouble began!

Trajectory chaining is essential for a robot that is behaving in a reasonably free form mode and performing general tasks in its workspace. These *chained* trajectory segments have targets sequenced as in figure 4.2, but the targets are visited one after the other without the respite of returning to a common home position. This technique caused the system to attempt a closed trajectory that must be repeatable to achieve stable trajectory formation. Unfortunately, using this mode of operations, ARTISTS's first results looked like figure 4.3. In that figure it is apparent that posture evolved over time to one that scarcely resem-



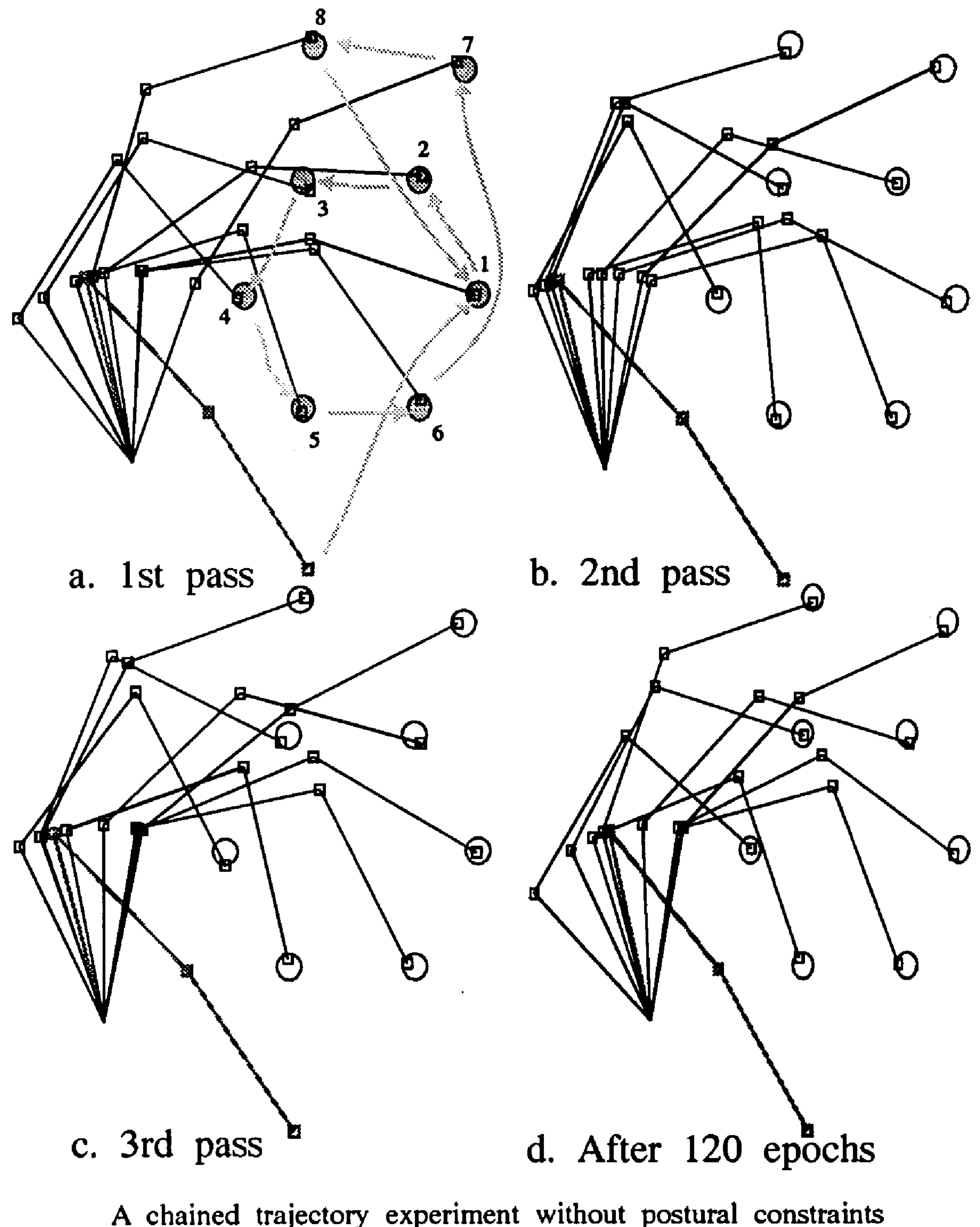
A sequential reaching experiment without postural constraints
genl = 64 quant = 4,8 memsize = 20K
Final memory vector count = 756

Figure 4.2: Sequential Reaching Without Constraints



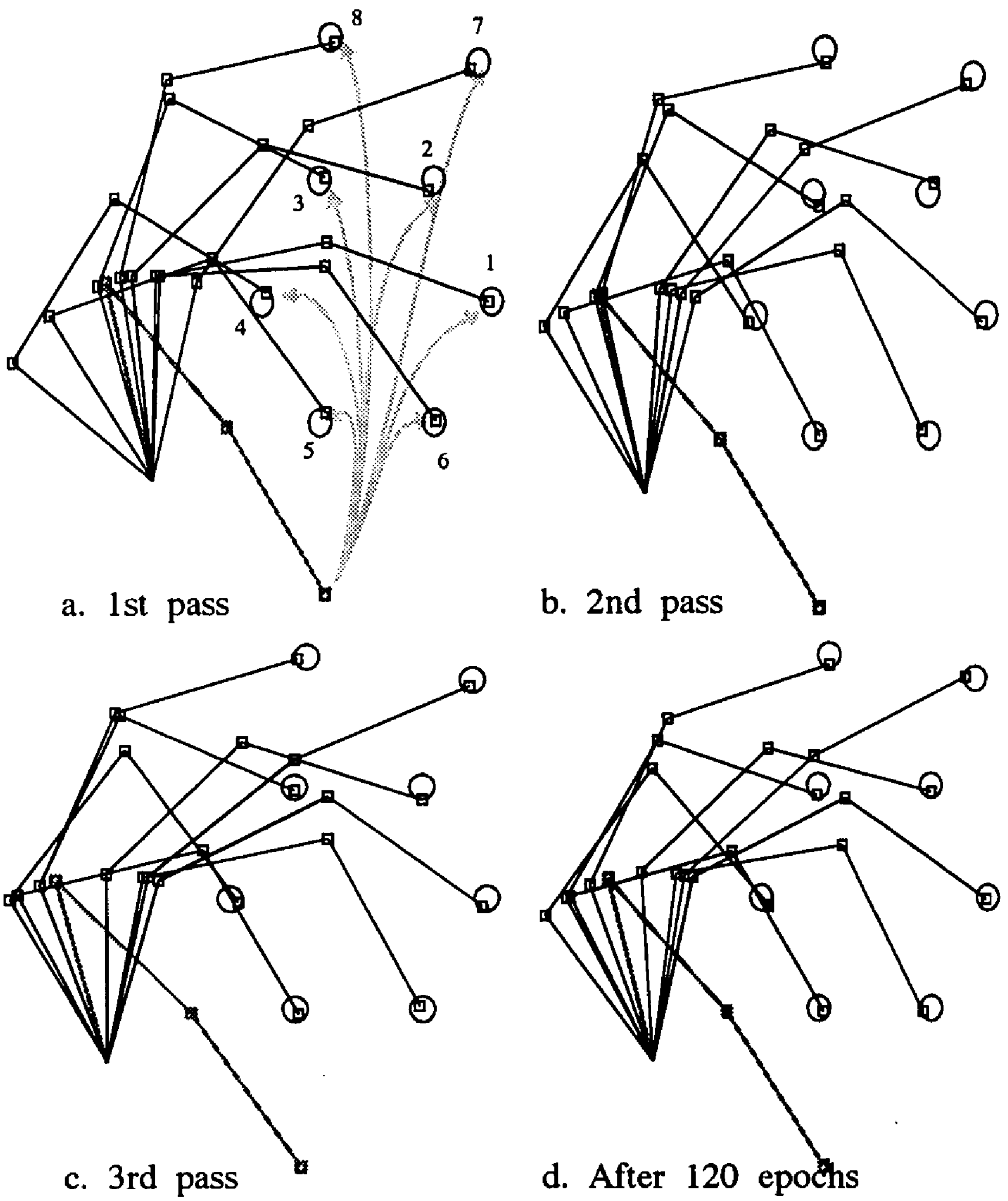
A chained trajectory experiment without postural constraints
genl = 64 quant = 4,8 memsize = 20K.
Final memory vector count = 2039

Figure 4.3: Chained Trajectories Without Constraints



A chained trajectory experiment without postural constraints
genl = 64 quant = 4,8 memsize = 20K
Final memory vector count = 1215

Figure 4.4: Chained, Posturally Constrained Trajectories



A sequential reaching experiment with postural constraints
genl = 64 quant = 4,8 memsize = 20K
Final memory vector count = 739

Figure 4.5: Posturally Constrained Sequential Reaching

bled the initial posture. In reaching from the home position to any one target, as in figure 4.2, no such evolution had been immediately obvious.

What I concluded from this was that when approached from the home position, the posture at the $i^{th}$ target was not necessarily the same one that resulted from target $i$ being approached from the $i\text{-}1^{st}$ target.

Panels a through c of figure 4.3 each show a single pass around the targets starting from the home position. In each case, the initial curvate posture had been more or less preserved from targets 1 through 4, with angle $\beta$ decreasing only slightly. As the trajectory proceeded through targets 5 and 6, the posture became noticeably different and by the time path $6 \rightarrow 7$ was executed, the posture was quite different, even though we might have expected a return to an earlier posture, since this path segment passed very near target 1.

I reasoned that if a stable repeatable posture existed in the closed trajectory, then stable trajectory formation might result. This is what happened in panel d: angle $\beta$ finally drifted down during 120 path segments with no return to any common grounded posture until it reached 10°, from which point no further drift was possible, because angle $\beta$ struck a joint stop there. Since angle $\beta$ became a constant there, then in a region around that posture, dimensionality of the search problem locally reduced to 2, a non-redundant situation, which guaranteed a unique trajectory solution in that neighborhood!

This grounded the closed trajectory and prevented further drift from occurring. In general, trajectory drift will continue until such a grounding instance occurs. Memory use will increase too, because state space location is a function of posture. If posture differs slightly each time a trajectory passes through a par-

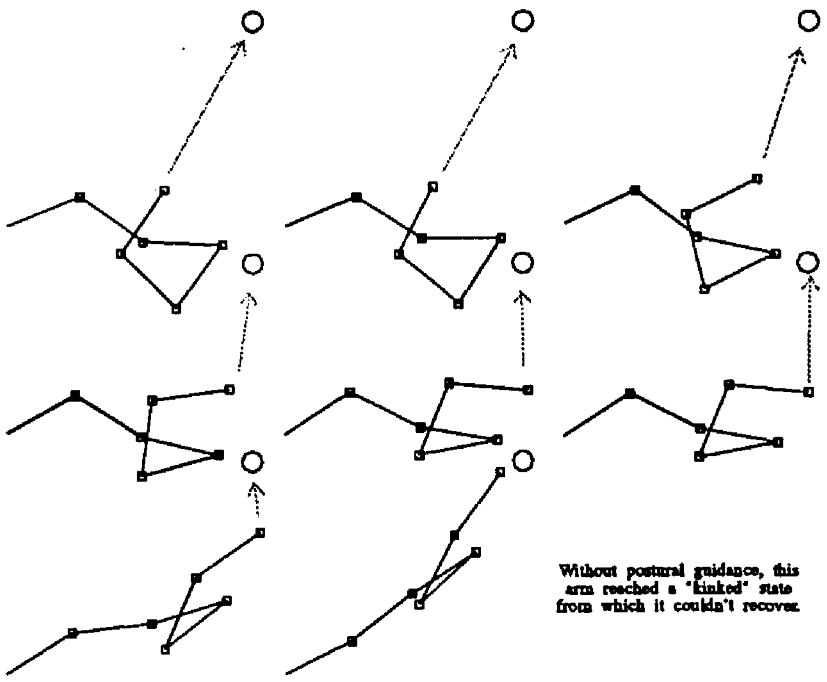

Without postural guidance, this arm reached a 'kinked' state from which it couldn't recover.

Figure 4.6: Kinking, an Extreme Case of Postural Drift

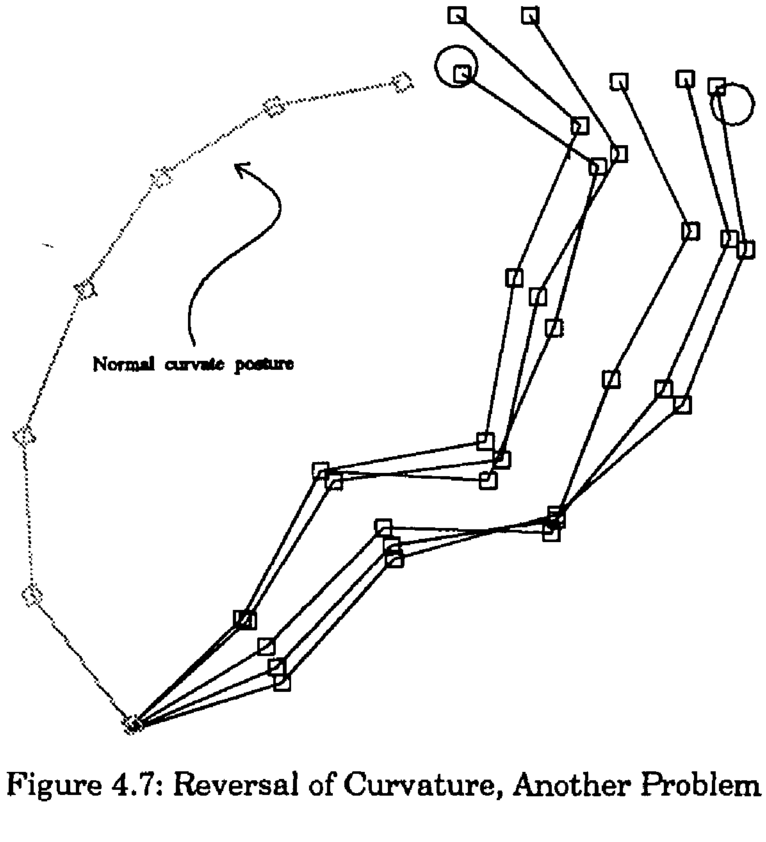


Figure 4.7: Reversal of Curvature, Another Problem

ticular target's neighborhood, the posturally generated address vector there will differ and so a slightly different set of CMAC weights will be adjusted to have non-zero values. So each pass around the targets, more CMAC weights will be non-zero until the trajectory stably repeats.

This interpretation was further reinforced by the following evidence:

- If an initial posture like that in figures 4.4 and 4.5 was used, the resultant chained trajectory always looked like panel d of figure 4.3 even during pass 1. This can be explained by observing that the new recumbent initial posture and the postures at targets 4 and 5 in panel d are similar.
- The amount of memory used by the unconstrained trajectories of figure 4.3 was large compared to that consumed by the unconstrained sequential reaching task of figure 4.2. This is reasonable because the sequential reaching task was grounded by the mechanism of starting at a common home position each time. The short trajectory from there to each target did not allow the opportunity for the posture to drift across large regions of state space.

The postural drift observed in these 4 figures became more problematic with longer kinematic chains. With more joints, there was more redundancy and ultimately situations like figures 4.6 and 4.7 occurred. In a real robot arm the kinking problem can not happen for obvious reasons, but the simulator I built was not capable of detecting this degenerate posture. It just became mechanically disadvantaged. In some cases, the kinking or reversal of curvature became extreme enough that without back track search capability, the simulation got stuck in a local minimum while seeking the next target and could not proceed.

I installed a postural constraint method, which will be developed in detail in section 4.6. Once that was perfected, figures 4.4 and 4.5 resulted, in which case smaller amounts of memory were consumed, and postures like figure 4.4 resulted, regardless of chaining or initial condition.

So it would seem that in all these cases, at some point there is a grounding of the posture, by the home position for the sequential reaching tasks, and by the terminal drift condition of the target 4 and 5 postures for the unconstrained chained trajectories of figure 4.3.d. These groundings locally reduced the degrees of freedom of the system

Altogether, the deficiencies inferred by the preceding experiments required the development of a stronger methodology. That stronger method involves attempting to ground the posture everywhere. This causes a reduction in dimensionality similar to that advocated by Hogan ['92]. (See section 4.6.1.)

### *4.4.1* ***Postural Feedback.***

Postural drift allows an arm to evolve the posture of its initial condition into a degenerate posture. Two factors that exacerbate this problem are trajectory chaining and longer kinematic chains, i.e., more joints.

ARTFORMS-1 operates open-loop with respect to posture and this is problematic. It results in postures drifting into mechanically disadvantageous configurations from which it may even be that no recovery is possible within the constraints imposed by the critic modules. Postural feedback is imposed by the postural constraint satisfaction discussed in section 4.6.

*4.4.1.1* ***Joint Stop Ratcheting.*** Joint stop ratcheting was another useful method of preventing degenerate postures, since even with postural constraints, kinking and reversal of curvature became a problem for longer kinematic chains. This technique simply involved variable joint stops that could allow the arm to start out in a degenerate posture, like the starting position of figure 4.22 on page 134, but once the posture opened up into one that was not in violation of strictly curvate joint stops and thus was within the range of allowed postures, the joint stops became effective.

Consider figure 4.20 on page 130. The initial posture was highly enfolded. Some joints were 10°, while other were > 180°. The ratcheting condition said that once the posture extended into a curvate form such that all joints (except the base, which is a free variable) were set such that $90° \leq \theta \leq 180°$ then 90° and 180° became hard lower and upper joint stops. The upper limit effectively prevented reversal of curvature and the lower limit (remarkably) prevented "kinking". The reason for this result is simply that the joints stops blocked any adaptive moves that descended the postural gradient in the "wrong direction". The reason for needing the 180° ratcheted stop was that even with postural constraint satisfaction making solutions non-redundant, I still had to worry about the kinds of postural switching redundancies mentioned in section 1.3.1.1. Whenever joints were near 180°, the alternate solution that caused reversal of curvature became as nearly likely as the one that favored the normal curvate posture.

*4.4.1.2* ***Observations About Compliance.*** There are implications for direct inverse systems brought to light by this notion of joint stop ratcheting that deserve mention. When the joint stops prevent incursion into a postural "forbidden-zone", if the robot repeatedly tried to enter that zone, excessive wear could result. But by using direct inverse modeling, the robot will never, under adaptive control, try to train a move into that forbidden zone. Since the robot can not do it, the direct inverse method can not train such a thing into the system. This means that the robot will, on average, tend to avoid the joint stops. I have observed that the arm shows no anomalous behavior upon encountering joint stops. The remarkable success of the 6 and 9 link arms in rapidly forming trajectories of curvate postured arms starting from very degenerately shaped initial postures bears witness to this state of affairs.

When a joint stop is encountered, one component of an incremental joint move vector will be decreased. This means that a move that was originally postulated as a unit length step in the work space will become shorter. The AGC mechanism will linearly scale that move up. (See section 3.2.4.1). It should be clear then that longer kinematic chains will behave more nicely upon encountering joint stop impingement because the linear scaling will be more near unity than for a joint obstruction of a shorter kinematic linkage.

## 4.5 *The ARTISTS Layer Preserved*

ARTFORMS-2 preserved *in toto* all of ARTISTS. The implementation of postural constraints and obstacle avoidance was added as a layer of additional training which merely added terms to the training of the LTM CMAC.

## 4.6 *Postural Constraints*

The postural feedback mechanism that was mentioned before is implemented by installing postural constraint satisfaction in the system. Some effects of these constraints are discussed.

#### *4.6.1* ***Postural Constraints Decrease Dimensionality***

The central motivation for use of the postural constraint is to make the redundant problem less redundant. Hogan ['92] argues that humans solve redundant kinematics problems by adapting to kinematic constraints which reduce the degrees of freedom of the system. Though not stated so succinctly, Jordan's work consists of constraint application that essentially does the same thing.

#### *4.6.2* ***Postural Constraints Increase Goal Directedness***

The postural constraints provide an immediate goal for the system to work toward. By assuring more continuity of postures along trajectories, it tends to force the direct inverse system to operate in state space locations similar to the locations of interest relative to what's being done at the time. In other words it tends to increase the overlap of adjacent receptive fields along the trajectory. (See section 4.8.2.6 on generalization slew rate, and section 5.2.2 concerning robustness of postural constraint training.)

#### *4.6.3* ***Postural Constraints Decrease Memory Saturation***

It was observed that for all experiments in which constrained arm moves were studied alongside unconstrained experiments, with all other aspects of the experiments being equal, the constrained system used less available CMAC memory than the unconstrained experiments.

The reason for this phenomenon is easily understood. During training, if the movements are constrained, then fewer kinds of different postures will be searched, and thus smaller regions of state space will be swept out during the search. Also the search will take less time, and it could be argued that since memory usage is a monotonically increasing function for CMAC, the longer early training takes, the more memory will be consumed.

### 4.6.4 *Development of Postural Constraint Equations*

The postural constraint equations will be developed by first stating the objective functions $\boldsymbol{F}(\theta)$ which are functions of the constrained angles (i.e. the ones other than the base angle). Next, a positive definite (in fact diagonal) $\boldsymbol{L}$ matrix is defined and the expression $\boldsymbol{F}^T\boldsymbol{L}\boldsymbol{F}$ is used to develop a least squares derivation to arrive at the software implementation of the gradient descent equations.

*4.6.4.1* ***The Objective Functions.*** The objective functions are just a set of linear constraints among the n joints. Each constraint is of the general form $k_i \cdot \theta_i - k_{i+1} \cdot \theta_{i+1}$. The objective of each constraint is the constraint equation $k_i \cdot \theta_i - k_{i+1} \cdot \theta_{i+1} = 0$.

For a normal curvate arm, all the *k*s would be equal, causing all the joints beyond the base to be more or less equal. The assumption that the absolute condition of equality is an ideal and may not be met within the training requirements of the system, gives rise to the "more or less" clause. For $k_i \neq k_j$, the two joints i and j are related by the ratio, $\frac{k_i}{k_j}$.

*4.6.4.2* ***Minimum Norm Derivation.*** If all the joints were constrained, including the base, the constraint equations generalize to the cyclic form $k_i \cdot \theta_i - k_j \cdot \theta_j = 0$, where $j = (i+1 \bmod n)+1$. This just adds the equation $k_n \cdot \theta_n - k_1 \cdot \theta_1 = 0$, for base angle $\theta_1$, and *n*th joint = $\theta_n$. In such a case, the system is overspecified, because it reduces the number of free variables to 1 (if, of course $f_i(\theta) = 0$, $(\forall\ i)$ could actually be satisfied, where $f_i(\theta)$ is the $i^{th}$ component, corresponding to joint *i*, of the vector function $\boldsymbol{F}(\theta)$). This is unnecessary and in fact undesirable, because if the constraints are always converged upon by gradient descent, the solution will never settle down, because eventually, for a fixed learning rate, a limit cycle will always be reached. On the other hand, if we heed Jordan's suggestion and use a declining rate to achieve a so called minimum norm solution for all these constraints, then ultimately we will have thrown out the postural feedback mechanism that this whole effort was all about! Clearly it is not desirable for on-line learning systems to have as a goal a system that operates open loop with respect to any important parameter. If the system were ultimately intended to reach stasis, and every parameter of the system fixed forever because we have exhaustively learned the system, that might be a good time to consider such a reduction to open loop operation, but that is not what we are trying to do here. At the risk of being tedious, I might suggest another spatially distributed parameterization to allow the learning rate for a minimum norm derivation to be tied to the experience level of the system as a function of state space location. (See section 3.3.1.2.) My efforts to implement minimum norm tended to reach a state of rather bad postural oscillation, which is what the above discussion predicted for a large constant learning rate, so the method that has as its goal an exactly specified system was finally chosen for ARTFORMS-2.

With this goal in mind, consider the following expression for the constraint equations:

$(IK - I_s K)\,\theta = \mathbf{0}$ where $I_s$ is the first superdiagonal identity matrix and $K$ is a diagonal matrix where $K_{ii} = k_i$.

In other words,

$$(IK - I_sK)\,\theta = \begin{pmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 0 & 0 & 0 & \dots & -1 \\ -1 & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & k_n \end{pmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdot \\ \theta_{n-1} \end{bmatrix} \qquad (4.1)$$

$$(IK - I_sK)\,\theta = \begin{pmatrix} k_1 & -k_2 & 0 & \dots & 0 \\ 0 & k_2 & -k_3 & \dots & 0 \\ 0 & 0 & k_3 & \dots & 0 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 0 & 0 & 0 & \dots & -k_n \\ -k_1 & 0 & 0 & \dots & k_n \end{pmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdot \\ \theta n-1 \end{bmatrix}. \qquad (4.2)$$

Now, define $L = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_i & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$ as a weight matrix for the constraints.

Setting any $\lambda_i = 0$ ignores constraint $i$. The complete set of constraint equations can be concisely expressed as

$$\mathbf{F}(\theta)L = (I - I_s)K\theta L = 0$$

$$= \begin{bmatrix} \lambda_1(k_1\theta_1 - k_2\theta_2) \\ \dots \\ \lambda_n(k_n\theta_n - k_1) \end{bmatrix} \qquad (4.3)$$

Now we must specify the L matrix. The base angle is set off as being qualitatively different in that it is grounded. So it seems a logical step to allow it to be a free variable and let the other *n-1* joints be related by **F**(θ)L.

Letting $f_i(\theta)$ be the components of $\boldsymbol{F}(\theta)$, then every $\lambda_i$ that contains $\theta_1$ should be set to 0, all others are set to the relative importance of the constraint to which they correspond. Therefore, $\lambda_i$=0 for i = 1 and *n*.

*4.6.4.3* ***Computing the Gradient of the Objective Function.*** This section describes the gradient of the objective functions and how to train the CMAC to incorporate an additional level of training to respond to it.

Constructing the least squares form of the functional (i.e. set of objective functions) as $\boldsymbol{F^T L F}$, we recall that the complete derivative of this form is

$$\frac{d}{d\theta}\left[\boldsymbol{F^T L F}\right] = 2\boldsymbol{F}L\frac{d\boldsymbol{F}}{d\theta}. \qquad (4.4)$$

So the gradient of the functional, $\nabla_\theta \boldsymbol{F} = 2\boldsymbol{F}L\frac{d\boldsymbol{F}}{d\theta}$, and adjustments to the joints to enforce the postural constraints will be: $\delta\theta = -\eta_c \cdot \nabla_\theta \boldsymbol{F}(\theta)$, where $\eta_c$ is a constraint learning rate not to be confused with the CMAC learning rate $\eta$. In fact, on the assumption that $\lambda_i = \lambda_j = \lambda$, for all $i,j \neq 1$ and $i,j \neq n$, then $\lambda$ can be factored out leaving $\delta\theta = -\lambda \cdot \nabla_\theta \boldsymbol{F}(\theta)$.

Now observe that since all the constraints are linear, from equation 4.3 we can derive the Jacobian matrix,

$$\frac{dF}{d\theta} = \begin{pmatrix} k_1 & -k_2 & 0 & \dots & 0 \\ 0 & k_2 & -k_3 & \dots & 0 \\ 0 & 0 & k_3 & \dots & 0 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 0 & 0 & 0 & \dots & -k_n \\ -k_1 & 0 & 0 & \dots & k_n \end{pmatrix} = (IK - I_sK) \qquad (4.5)$$

So, substituting equation (4.5) into equation (4.4), we find that

$$\nabla_\theta = \lambda \begin{bmatrix} 0 \\ -k_2 f_2 \\ k_3 f_2 - k_3 f_3 \\ \dots \\ k_{n-1} f_{n-2} - k_{n-1} f_{n-1} \\ k_n f_{n-1} \end{bmatrix} \qquad (4.6)$$

Substituting $f_i = k_i\theta_i - k_{i+1}\theta_{i+1}$ into equation (4.6), we finally get the form,

$$\delta\theta = -\lambda\nabla_\theta F = -\lambda \begin{bmatrix} 0 \\ -k_2(k_2\theta_2 - k_3\theta_3) \\ k_3(k_2\theta_2 - 2k_3\theta_3 + k_4\theta_4) \\ k_4(k_3\theta_3 - 2k_4\theta_4 + k_5\theta_5) \\ \dots \\ k_{n-1}(k_{n-2}\theta_{n-2} - 2k_{n-1}\theta_{n-1} + k_n\theta_n) \\ k_n(k_{n-1}\theta_{n-1} - k_n\theta_n) \end{bmatrix} \qquad (4.7)$$

In the implementation of equation 4.7, note that the leading $k_i$ in each $i^{th}$ term of the $\delta\theta$ vector simply provides an additional weighting on each of the terms in the vector. It is unclear at this point whether it would matter if the factored out leading coefficients were all set to unity, leaving only the ones inside the parentheses variable. Experiments seemed to indicate that this would not matter, but the extra computational burden of leaving them all variable is so minor that we may as well stay with exactly equation 4.7.

*4.6.4.4* ***Applying the Gradient of the Objective Function.*** In this section, the actual method used to incorporate the system's training is discussed. In figure 4.1, page 81, there are 3 training sources impinging on the direct inverse CMAC: direct inverse, postural and smoothness. These are listed here in order of application. The method used is to train each separately. The nature of the delta rule (first seen in figure 3.8) shows each training instance to be a summation into the direct inverse model. Jordan ['88] and others describe constraints used in gradient descent training being arbitrated by simply forming a training term that is a sum of constraint terms. Adding new constraints means adding new terms to the negative gradient expression. This is equivalent to the separate application of each training step as discussed below.

The smoothness constraint discussed in section 4.7 was never actually implemented. These experiments worked adequately without it, but its inclusion may have shortened the training time required for long kinematic chains.

*4.6.4.5* ***Step 1: the Direct Inverse Training Step.*** The following discussion is embodied in the C file try2lern.c (which is excerpted in Appendix F). As a first principle, the delta rule is reprinted (and paraphrased) from figure 3.8:

$$\underline{\Delta w} = \frac{(\,\underline{\Delta\Psi} - \underline{\Delta\hat{\theta}}\,)\,\eta}{C} \qquad (4.8)$$

This rule is applied twice. The first application is the direct inverse training step, with $\underline{\Delta\Psi} = \underline{\Delta\theta}$, where $\underline{\Delta\theta}$ is is the observed change in the robot's joints and $\underline{\Delta\hat{\theta}}$ is the direct inverse estimate already present in the CMAC at the current state space point. This operating point is defined by the input vector, inp_vec[ ], which is used to activate the robot to generate this change in joint angles. $\Delta w$ is automatically generated and added to the weights of the receptive field by a call to the function, learn(int cmac_id, int inp_vec[ ], int delta_joint[ ], int eta), provided by UNH_CMAC. ( delta_joint[ ] *is* $\underline{\Delta\theta}$ during training and $\underline{\Delta\hat{\theta}}$ after lookup.)

*4.6.4.6* ***<u>Step 2: the Postural Constraint Training Step.</u>*** In the second, or constraint satisfaction step, the delta rule is applied to cause the learned $\underline{\Delta\hat{\theta}}$ to be adjusted to favor the postural constraint satisfaction. In this step the negative gradient of $\boldsymbol{F}$, $\delta\theta$, is used so $\underline{\Delta\Psi} = \underline{\Delta\hat{\theta}}+\underline{\delta\theta}$. The details of the second training step involve recalling $\underline{\Delta\hat{\theta}}$, given the same inp_vec[ ] as at the just completed direct inverse training step, via a call to the UNH_CMAC function rembr(cmac_id, inp_vec, delta_joint). Now delta_joint[ ] is modified by a call to the function SatisfyPosturalConstraints(0). This incorporates equation 4.7. For now, just consider kv[ ] to be the diagonal of the constant $K$ matrix of that equation. In section 5.2.2, this vector becomes non-constant, so the code that manipulates kv[ ] may become clearer after studying that section. The purpose there, as will be seen, is obstacle avoidance.

So now the $\underline{\Delta\theta}$ vector is modified to favor the arm posture embodied in $K$ and equation 4.7. That is $\underline{\Delta\theta} = \underline{\Delta\hat{\theta}} + \underline{\delta\theta}$. All that remains to complete the job is a call to learn( ) just like the one in section 4.6.4.5 which trains with the new $\underline{\Delta\theta}$ as exemplar. A similar sequence of steps could be performed to learn the temporal constraints discussed in section 4.7.

#### *4.6.4.7* ***Robustness of Constraint Training With Respect to Learning Rate.***

The system was not overly sensitive to the value of $\lambda$ discussed above, which contrasts with most gradient descent based methods in the past. This issue is discussed further in section 5.2.2.

## *4.7* ***A Discussion of Temporal Constraints***

It has been previously argued in sections 3.4.6 and 3.4.7 that the overlapping receptive fields CMAC provides neighboring trajectory steps imposes an implicit smoothness constraint on trajectories. It is a simple extension of the system (not implemented at this time) to add the *explicit* smoothness constraint shown in figure 4.1 and 5.1 on pages 81 and 143, respectively. This would entail saving the "last" incremental joint move learned during the previous trajectory step and using it as a training exemplar at the current step. The learning rate used for this training instance would be like a momentum term parameter. A large learning rate would prejudice the system to strongly favor the same action as was recently taken. A stiffness term can be added by training an incremental joint move that favors the *posture* of the previous time step rather than the *move* of the recent step. Neither of these are implemented, but would make interesting future work.

Sutton's temporal difference method applies such temporal constraints, but requires back-propagation of errors through an analytical[1] model. There is a variation on the temporal difference method I call *n*th order temporal lookahead, that may be useful to parameterize systems capable of performing backtrack search in a reflexive manner. Ideas concerning this as future work are discussed in section A.1.1.2.

1 A multilayer perceptron is an "analytical" model of sorts, though not *directly* a model of the plant.

## 4.8 Experimental Results

This section describes the method of selection of the system gross parameterization, discusses the error metric formulas, and makes conclusions about the adequacy of the ARTISTS system with respect to robustness from a number of different vantage points

### 4.8.1 Selection of Adequate Parameters

This section discusses how the gross system parameterization (i.e. settings for generalization and quantization) were evaluated to find nearly optimal parameterizations.

*4.8.1.1* ***Definition of Gross Parameterization.*** A system parameterization includes all possible parameters that characterize the physical system and the problem it solves. Among these for an articulator are number of joints, linkage lengths, and even the adaptive values in the neural networks, which are viewed as fine grained variable parameterizations. Much of this dissertation talks about a spatially distributed parameterization stored in the latter. In order for the networks to provide a reliable mapping, the network's gross parameterization must be appropriate. This parameterization is defined as an ordered quadruple, $\mathbf{P} = (C, q_h, q_j, M)$, where $C$ = generalization, $q_h$ = quantization of the hand space target direction vector part of the input vector, $q_j$ = quantization of the joint components of the input vector, and $M$ = the physical memory size, in vectors. It seemed highly appropriate to have different values of $q_h$ and $q_j$ but probably unnecessary to distinguish the individual joints' quantizations one from the other. The directed search used to find an adequate parameterization involved about 250 experimental setups. Varying quantization on a joint by joint basis would have had an exponential effect on that search.

*4.8.1.2* ***Memory Size.*** The physical memory size is clearly an important system parameter. Large memories of $> 20{,}000$ vectors seemed to always be adequate. Small ones of $< 5{,}000$ vectors usually gave poorer results, but it is important to pick a minimal adequate memory size based on the problem, so data has been collected over a wide range of memory sizes. A consideration of innate noise that is related to memory size is discussed in section 4.8.7.

### *4.8.2* ***The Error Metrics***

During execution of trajectory formation for an 8 target, repetitive trajectory ensemble for a 3 joint redundant arm, several error metrics were tracked. These data were written out to files during trajectory formation, at the end of each path segment. The experiments all ran for 1000 path segments.

Each metric was computed as a root mean square or absolute summed value as appropriate, and scaled by dividing by the path segment length, so that the error function became a density function over the trajectory. Each was also scaled and offset so the error functions could be plotted afterwards on a CRT for visual evaluation. A comprehensive visual review of all the data allowed several settings to immediately be disqualified. Beyond that, the visual inspection method became too ambiguous, so a batch evaluation method was devised.

The data were then all assumed to eventually reach steady state. Under the steady state assumption, each error metric, for each experiment, had its mean and variance computed over the last 100 path segments.

All trials that resulted in a non-zero heuristic step density after the first 100 epochs were disqualified. The means and variances were entered into a spreadsheet, so that all 250 experiments could be evaluated. The importance of each error metric was weighted and the means and variances of the error metrics were sorted in turn for each metric. The top 11 experiments for each ranking were selected and the conjunction of the top ranked sets were subjected to a weighted sum test to determine a best parameterization for all the experiments. Details of the rankings are discussed in section 4.8.4.1.

From this final list emerged 4 candidate parameterizations, $\mathbf{P}_1$ = (32,4,8,*M*), $\mathbf{P}_2$ = (32,8,8,*M*), $\mathbf{P}_3$ = (64,4,4,*M*) and $\mathbf{P}_4$ = (64,4,8,*M*). Next, a sequence of experiments was run in which the means and variances for each $\mathbf{P}_i$ was determined for a wide range of $M$. The plots in figures 4.8, 4.9, 4.10, and 4.11, plot several important metrics, and the important point to note is that in general the system is robust with respect to physical memory size, with poorer performance for smaller memory size, and better for large. However, for reasonable memory sizes > 10,000 vectors there is virtually no difference in the performance metrics for a good parameterization (like $\mathbf{P}_4$, for instance).

Figures 4.8 and 4.11 seem to be problematic, though. In these figures there is a curious dependence on memory size. An increase in physical memory size should in all cases make the average error metrics go down. But these parameterizations did not conform strictly to that rule. Section 4.8.9 will discuss in detail an explanation of this phenomenon.

Below are detailed definitions of the error metrics used to evaluate the adequacy of gross system parameterizations.

*4.8.2.1* ***Postural Error Density.*** This metric computes the summed difference between the posture at each step and what is desired. In other words the RMS sum of the postural constraint objective functions. The factor of 1/10 compensates for the $1/100°$ scaling of the input mapping.

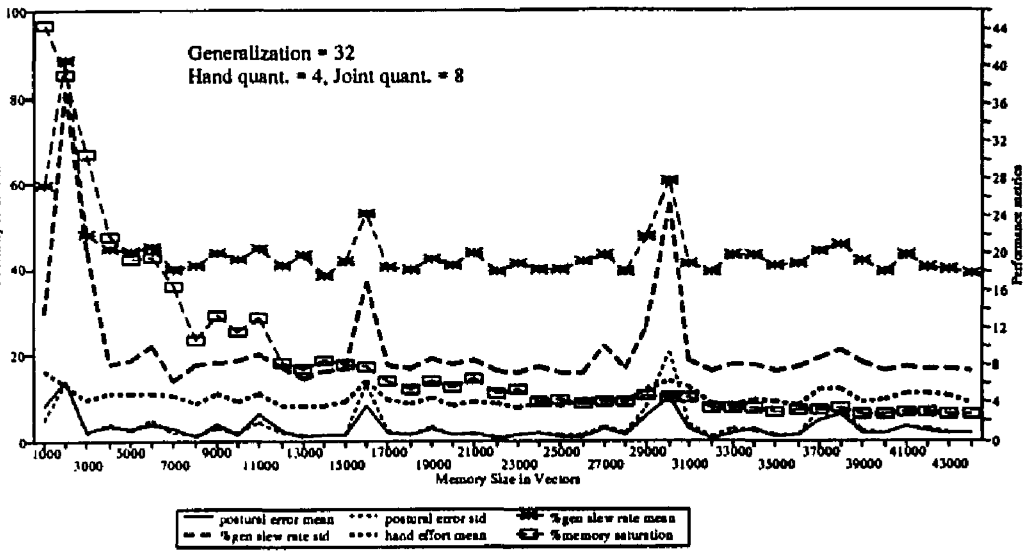


Figure 4.8: Anomalous Dependence on Memory Size
for C = 32

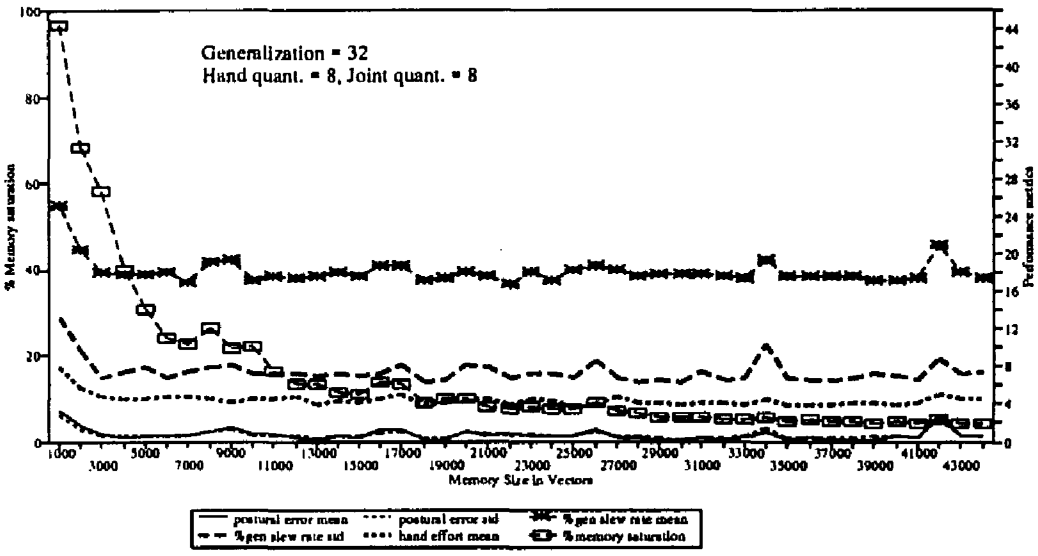


Figure 4.9: Robust Dependence on Memory Size
for C = 32

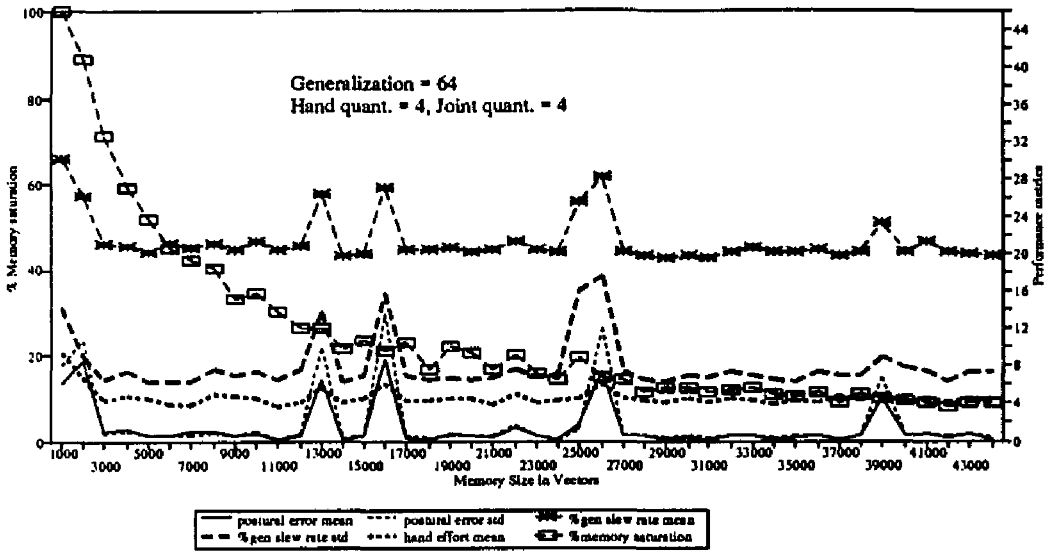


Figure 4.11: Anomolous Dependence on Memory Size
for C = 64

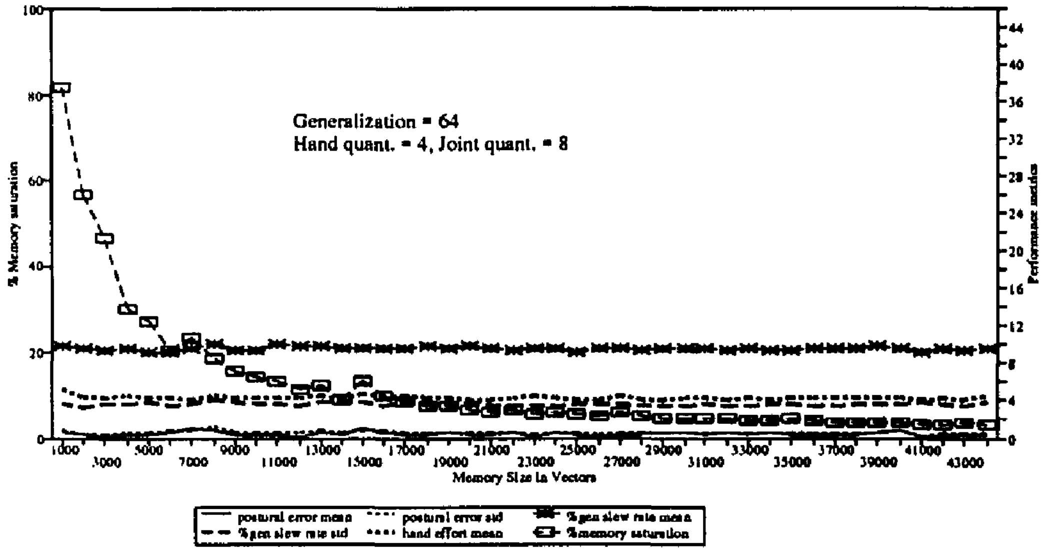


Figure 4.10: Robust Dependence on Memory Size
for C = 64

$$e_p = \frac{\sqrt{\sum_{path}\left\{\sum_{j=1}^{n-1}\left[\left(\frac{\theta_{j+1}}{10}\right)^2-\left(\frac{\theta_j}{10}\right)^2\right]\right\}}}{pathlength} \qquad (4.8)$$

*4.8.2.2* ***Joint Effort Density.*** This metric measures the RMS value of the total joint angle increment at each step.

$$e_j = \frac{\sqrt{\sum_{path}\left[\sum_{i=1}^{n}(\Delta\theta_i)^2\right]}}{pathlength} \qquad (4.9)$$

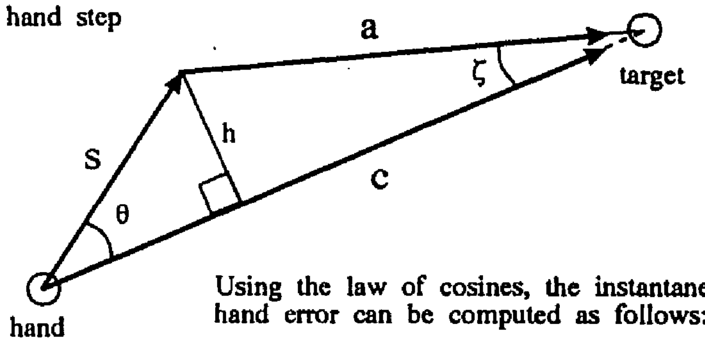
---

$$e_h = \frac{\sum_{path} h_i}{pathlength} \qquad (4.10)$$

a = new distance to target

c = old distance to target

s = hand step



Using the law of cosines, the instantaneous hand error can be computed as follows:

$$\theta = \cos^{-1}\left(\frac{s^2+c^2-a^2}{2\,sc}\right) \qquad h = s\cdot\sin\theta,$$

and all these quantities are readily available as a side effect of the execution of the simulation.

Figure 4.12 Computing Hand Error

*4.8.2.3* ***Hand Effort Density.*** In this metric, the law of cosines is applied to the desired and actual hand move vector at each step. The perpendicular displacement of the hand from the desired rectilinear path at each step is summed. Straight summing is sufficient here because this value, $h$, is always positive. This is a measure of the curvature of the trajectory. Figure 4.12 describes the derivation of hand effort and the instantaneous hand error, $h_i$ which is the perpendicular deviation of the actual hand move from the desired hand to target vector.

*4.8.2.4* ***Step Size Density.*** The steps in work space were intended to be of unit length. The actual length was computed and its RMS sum was stored. This value is computed as in equation 4.11.

$$e_{stp} = (\#\ \text{of steps}) \div \text{pathlength} \qquad (4.11)$$

This value should be nominally 1.00, though in practice it had a value of slightly more, and it did not turn out to be a reliable measure of performance, because it measured just what the heuristic step generator dictated and an automatic gain control mechanism was built in which scaled every step to within some tolerance of 1.00. This measure then was always consistent and seemed to be more or less independent of the goodness of fit of the trajectory to the trajectory formation goals.

*4.8.2.5* ***Heuristic Step Density.*** The total number heuristic steps were counted on each path and the count was divided by the path length. This metric should go to zero in the steady state, and did so for all successful experiments.

*4.8.2.6* ***Generalization Slew Rate Density.*** This is a difficult metric to describe. It is a measure of how much overlap there is between the receptive fields of adjacent trajectory steps. If the slew rate is 0%, it means that receptive field is the same for this step as for the last. If the slew rate is 100%, the two steps have exactly disjoint receptive fields. Slew rates of > 100% indicates that the receptive fields are even farther apart in the state space. Equation 4.12 describes this quantity, with $v_i^j$ being the $i^{th}$ component of the scaled integer input vector for the $j^{th}$ step in the trajectory. $q_i$ is the $i^{th}$ component of the quantization array; larger values of $q_i$ denote coarser codon representation.

$$\frac{e_g = \max\limits_{j \in path}\left(\sum_{i=1}^{n+2} \frac{\lfloor v_i^j - v_i^{j-1} \rfloor}{q_i}\right) \times 100}{C} \qquad (4.12)$$

*4.8.2.7* ***Memory Saturation.*** This metric is simply the cumulative memory usage of the CMAC. The measure is provided by the UNH_CMAC software. The value returned is the count of the non-zero-valued memory vectors in the physical memory allocated to a particular CMAC. This gives an approximate measure, $U$, of the number of the physical memory vectors actually used. A discussion of the inaccuracy of this metric is in section 4.8.10.2.

The value $u = U/M$ is the degree of saturation of the CMAC, and it has been determined by our cumulative results in the Robotics Lab that when $u$ is small, CMAC function approximation is good. When $u > u_{max}$ for $u_{\max} \approx 0.2$ or 0.3., performance suffers. This subject is discussed at greater length in section 4.8.8.

### *4.8.3* ***Discussion of the Gross Parameterization Results***

Appendix E, page 210, contains plots of selected experimental results. Figure E.0 contains plots of all the important error metrics for a single experiment, with parameterization **P** = (64,4,8,20000). In figure E.0 there is a set of plots for an experiment with the same parameterization but with each path segment repeated 5 times before the next path segment is executed. The effect of this is that it spreads the plots out left to right to show the near periodicity of certain error metrics more clearly.

The original reason, however, that I contrived experiments of the latter type was that I reasoned: if a path segment could be executed over and over again before a new path segment was tried, the result would be less postural drift and a faster learning of the overall trajectory. As is discussed in section 4.8.9, this turned out not to always be true, but by the time I had realized this, the selection of a "best" parameterization was completed. Re-running all those experiments with non-repeating path segments was deemed not worth the effort.

The multi-trace plots of Appendix E show clearly the rapid convergence of the error metrics upon which were based the selection of gross system parameterization. Some selections of memory size, generalization parameter and quantization were much better than others at finding a solution rapidly and reliably.

Approximately 250 experiments were crafted as described in section 4.8.2. After the data was collected in a series of files (one set for each experiment) these files were analyzed based on a steady state assumption for the last 100 path segments, and the following set of observations were made:

### *4.8.4* ***Details of the Ranking of Parameterizations***

*4.8.4.1* ***The First Order Ranking.*** 8 separate rankings of the means and variances of all the 250 experiments were made using the sort records feature of the Quattro-Pro spreadsheet program. Each ranking was based on a different error metric.

*4.8.4.2* ***Gross Eliminations Based on Heuristic "Need".*** Any parameterization whose experiment had non-zero heuristic step density during the terminal or "steady state" phase of the experiment was disqualified. This removed 7 experiments from the list. Of the remaining experiments, the topmost 11 experiments were selected, in each of the 8 rankings.

*4.8.4.3* ***Strong Correlations Among the Metrics.*** It was noted that hand effort, joint effort, memory usage (and percentage memory usage) and postural error were strongly correlated. The same set of experiments tended to be the best experiments in each ranking, as they were ranked, metric by metric.

### *4.8.5* ***Subjective Weightings***

The following reasons were used for setting the weights attached to the ranks in each of the 8 lists as relatively high, low or unimportant. The weight was a number between 1 and 10.

*4.8.5.1* ***Step Size.*** Step size density was given a low weight of 1 because it appeared to be only weakly correlated to success.

*4.8.5.2* ***Percentage of Memory Used.*** Memory saturation was an ambiguous measure of success, because, by using large physical memory (even when it was not necessary to do so), this quantity could look good even when other measures were mediocre or terrible.

*4.8.5.3* ***Number of Memory Vectors.*** This metric was more reliable, because since all setups performed the same task, one might reasonably expect similar memory usage as an absolute vector count.

For instance, consider a setup not coarse coded enough, like $\mathbf{P}_u$ = (32,2,2,20000). 19817 vectors were used, as compared to 1500 to 2000 vectors for the best performers, and its performance was, as expected, bad. On the other hand, $\mathbf{P}_o$ = (32,16,16,5000) resulted in only 699 vectors being used, but that setup had twice the mean and variance of the best hand effort and postural error statistics achieved. $\mathbf{P}_o$ approaches (rather distantly of course) the limiting case of storing all the information in the same location, degenerating the CMAC into a simple integral controller, which by itself we would not expect to do a good job at this task.

Though much can be inferred by vector count, its ambiguity gave me cause to give it a low importance, but one higher than either percentage memory saturation or step size density.

*4.8.5.4* ***Generalization Slew Rate.*** The assumption that $\mathbf{P}_o$ "over-generalized" is confirmed in that case by the fact that the generalization slew rate for that experiment was the minimum value of all the experiments.

At the other extreme, when the quantization is too small, under-generalization occurs. $\mathbf{P}_u$, for instance, showed a generalization slew rate of >500%, and performed worse than table lookup. This setup did not have a sufficiently large memory space to learn at all. With table lookup, learning should occur, albeit slowly. With 99.1% memory saturation, however, *this* setup was clearly distracted from accomplishing learning by an excess of hashing collisions. In fact, as is discussed in section 4.8.8, we might reasonably conclude that such a system would approach an expected value of hashing collisions that is 90% or more of its receptive field size every time it maps an input, because the large generalization slew rate indicates a considerable reduction in the correlation between adjacent lookups along the trajectory.

Although generalization slew rate is important, neither a least nor a maximal value infers a particularly good parameterization, so slew rate was not included as a ranking metric; rather, all experiments with generalization slew rates > 30% were eliminated. Appendix C contains some analysis to validate that the observed slew rate of < 20% for the best parameterizations is a reasonable and necessary result. This result is important for the conclusion in section 6.2, concerning the view of ARTFORMS as an extension of traditional linearization methods.

*4.8.5.5* ***Joint Effort.*** Joint effort was given a relatively low weight, because, as has already been mentioned, it did not correlate well with success. It can be observed in figure E.0, that joint effort is more or less a periodic function that develops early and independently of the other metrics. The reason for this is that joint effort is dictated by the heuristics generator. It always postulates displacements selected with the goal of unit length hand moves. Thus, when the arm is extended, the joint effort is small, and when the arm is tightly enfolded, the joint effort is large, all based on the moment arm of the articulator. This extension and retraction rhythm is clearly visible in the plot. We must also keep in mind that the AGC mechanism (see section 3.2.4.1) that scales all attempted moves to generate unit length hand moves is a tight loop feedback mechanism that ensures joint effort will be a mediocre measure of performance relative to learning. So joint effort is given a rather small weight, similar to the weight assigned to memory vector count.

*4.8.5.6* ***Hand Effort and Postural Error.*** These two metrics were given the highest weights, because they represent direct measures of what the direct inverse training step and the postural constraint training step are actually learning. As is argued in section 4.4, the latter error measure ultimately measures the stability of the system's storage and retrieval capabilities when the problem is redundant.

### *4.8.6 The Final Conclusion of the Rankings*

The net result of these ranking tests was that no matter what values of the weights I assigned, within the general guidelines of the last 6 subsections, the following parameterizations always won: $\mathbf{P}_1$ = (32,4,8,M), $\mathbf{P}_2$ = (32,8,8,M), $\mathbf{P}_3$ = (64,4,4,M) and $\mathbf{P}_4$ = (64,4,8,M).

### *4.8.7 Innate Robustness in the Presence of Noise*

Figures 4.8 through 4.10 clearly show that there is a robustness in this methodology that, subject to the caveats in section 4.8.9, can be seen as a function of memory size. For any reasonable parameterization (especially $\mathbf{P}_4$) the only poor performance this methodology shows is selection of a too small memory space, and that once a threshold memory size is achieved, the error metrics remain quite robustly low for all memory sizes. There is an inherent principle hidden in these results that I will try to articulate.

Jordan ['88] showed that his forward modeling method was robust in the presence of noise by injecting random noise in the forward model after trajectory formation had converged. The result was that it still performed (though with noticeable degradation of performance especially relative to posture). Contrarily, in my methodology, I contend that there is an innate and quantifiable level of noise that can be observed directly as texture in simpler CMAC mappings like figure 3.5 on page 69. This noise level is of course related to hashing collisions; it is quantifiable using the methods of Appendix B, and it is inherent and unavoidable. The fact that from an external observation the performance of most well parameterized experiments with memory sizes above 5000 vectors were virtually indistinguishable one from the other shows that this methodology is quite immune to this inherent noise. Only when the noise is turned up to very high levels does the mapping suffer as in the far left hand side of the plots of figures 4.8 through 4.10. In an investigation of the nature of the heuristic and training critics, I further disturbed the system by injecting noise into the internal model of the plant used as a predictor (not as a control effort source) and it showed graceful degradation in the face of that noise,too. (See section 4.8.12.)

### *4.8.8* ***A Linear Upper Bound on Hashing Collision Damage***

To a limited extent, the probabilities of hashing collisions given prior training, i.e. how badly hashing collisions are likely to disturb old data by the introduction of new data, can be analyzed. Simple analytical methods are developed in Appendix B. In that Appendix, I show that in fact the expected value of the number of constituent points in a receptive field that point to already prior used data is strictly a function of memory saturation. The surprising result is that the function is linear. A proof by construction therein shows that this value

$$E_C(u\,,C) = \sum_{n=1}^{C} (1-u)^{C-n} \cdot \binom{C}{n} \cdot u^n \cdot n,$$

is really just $E_C(u, C) = u \cdot C$. This represents a very loose, worst case situation for random input sequences. Experimental results in this study of course represent highly correlated sequences of data, and so of the "collisions" incurred by adjacent mappings along a trajectory, most are intentional. In fact, for $\mathbf{P}_4$ with its maximum generalization slew rate falling in the range of $< 10\%$, then 90% or more of those collisions are probably intentional, due to the large overlap of adjacent receptive fields. This signifies that the upper bound, $u \cdot C$, is extremely loose, and the actual useful memory capacity may be *profoundly* large. Section 4.8.11.1 indicates that 44,000 vectors under the $\mathbf{P}_4$ parameterization is more than sufficient for virtually any set of trajectories definable for a planar arm of up to 6 joints.

## *4.8.9 A Surprising Dependence on Physical Memory Size*

In section 4.8.2 I mentioned some interesting points concerning the plots of figure 4.8. The observation that it was possible to see a significant, measurable degradation of performance in a system by *increasing* memory size was somewhat disturbing. A new series of experiments was devised, in which the gross parameterization was $\mathbf{P} = (32,4,8,M)$, for M ranging over all integer values over the range (23990,... 24010). The results of the experiments indicate a possible dependence of performance metrics on memory size that had far smaller granularity than was expected (granularity = 1 vector out of 24,000).

The observed performance disturbance was of low magnitude, but it points out a possible source of significant error for systems that are trying to be very frugal with respect to physical memory allocation. Once the phenomenon was observed, I started probing that experiment to see if I could isolate the cause of the measurably different trajectories. It turned out that the performance problem was easily observed in this experiment.

Figures 4.13 and 4.14 show the behavior that gave rise to this error. All experiments in the series were subject to equiangular joint constraints. But, in moving from target 7 to 8, joints 2 and 3 showed noticeable constraint violation. The error corrected itself in going from target 8 to 1. Along the paths from target 3 to target 4 and 4 to 5, another minor but noticeable postural switching occurred.

This error appears to contradict our ideas of CMAC memory capacity until we realize that hashing collision error predictions are based on expected values. So over vast numbers of training instances the effect of hashing collisions should be very small if $u$ is small. Consider that we have an actual instantiation of a CMAC trained with the information of this experiment. There *are* hashing collisions. When the input mapping algorithm executes, it generates target addresses in the physical memory space by randomizing each of the $n+2$ coordinates of the input vector using a *fixed* randomizing lookup table. The $n+2$ coordinates are then added together to generate a virtual address. Now that virtual address, $A_v$ is limited via a modulus operation $A'_v = A_v$ *modulo memory size.*

The effect of the modulus operation is that it "folds" the virtual memory space back on itself however many times the virtual memory space is divisible by memsize. This folding operation defines $m$ partitions, where $m = \left\lfloor \frac{S_v}{memsize} \right\rfloor$.

Now any hashing collisions that exist in the mapping are fixed and act like im-

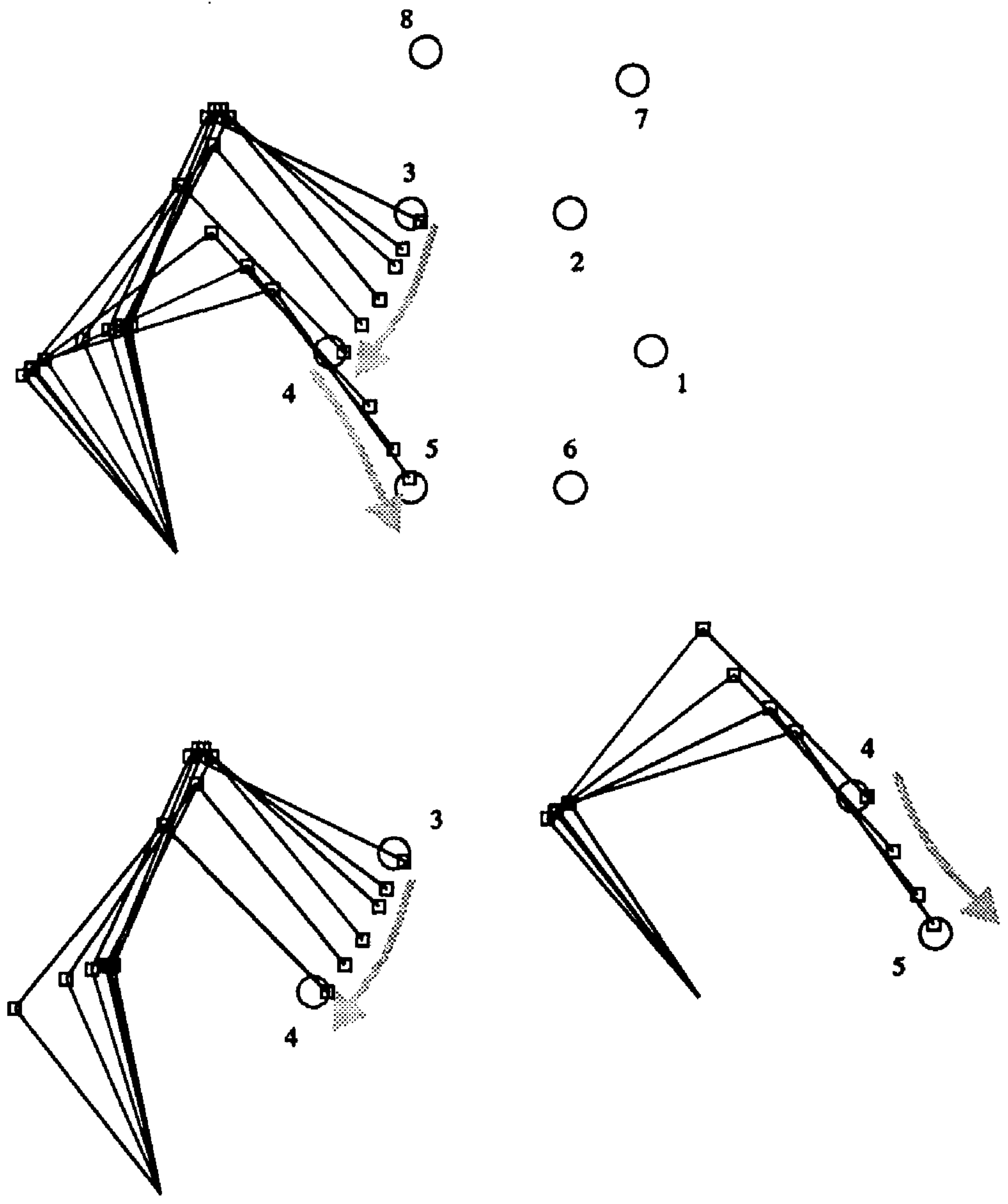


Figure 4.13: Anomalous Behavior for Memory Size of 24,002 Vectors

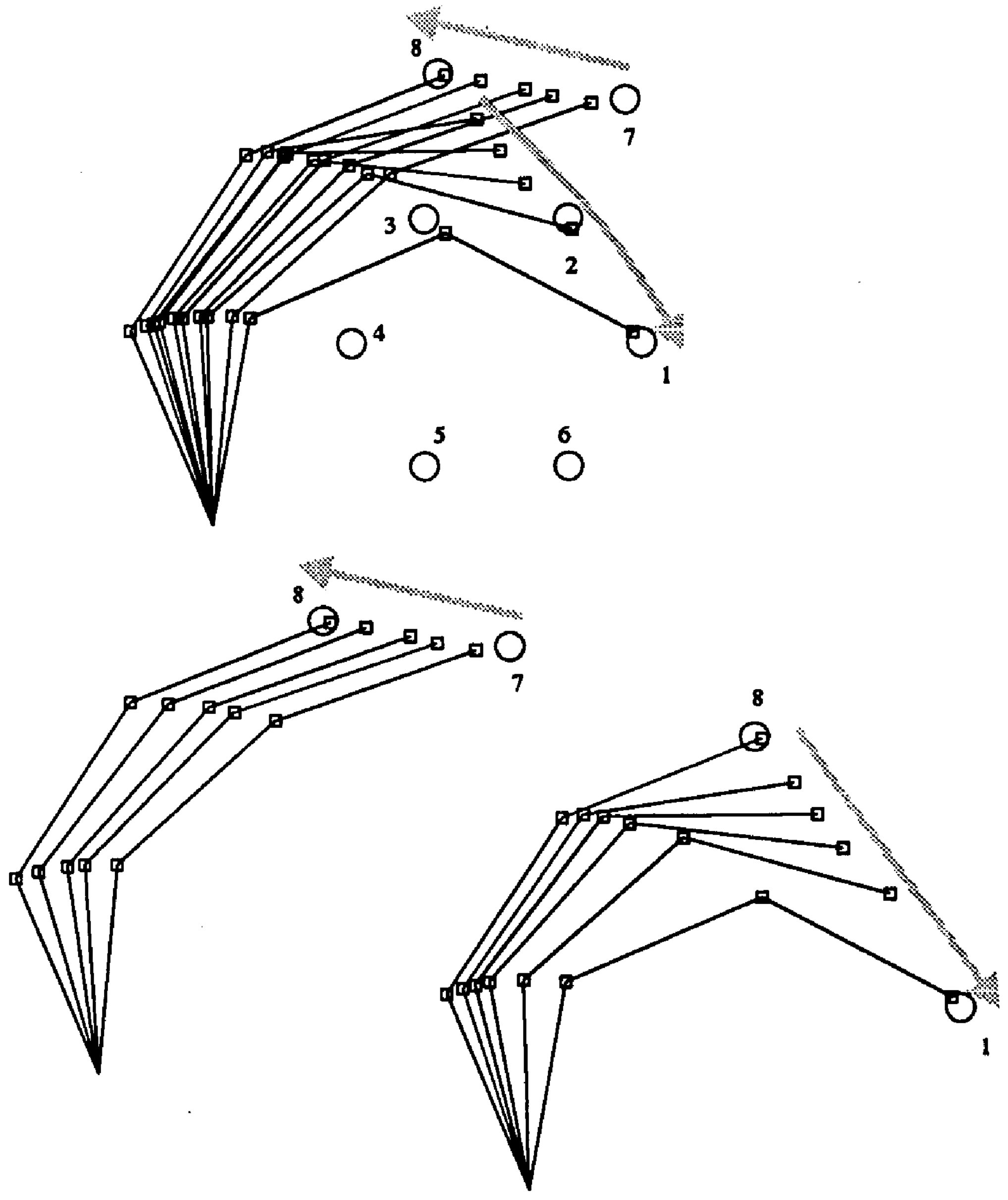


Figure 4.14: Anomalous Behavior for Memory Size of 24,002 Vectors

pulse functions in the state space. If one or more hashing collisions occur at a point exactly *k* codon units up from a fold caused by the modulus, but in different partitions of the virtual memory space, they will potentiate one another's effect. If the value of memsize is changed by one vector, that potentiation will disappear, because those hashing collisions will no longer be coincident in the physical memory.

The error seen in one of the "bad" experiments, e.g. memory size=24002, corresponding to the plots of figure 4.15 decreases significantly for the following actions:

- move targets 7 or 4 even by a small amount (figures 4.17 and 4.18 ),
- change the number of memory vectors by +/- 1 vector (figures 4.15 through 4.18) or
- change from chained trajectories with repeated path segments to ones without path segment repeating (figures 4.16 and 4.18 ).

The first of these action serves to move the trajectory away from a state region damaged by hashing collisions. The second action causes these hashing collisions to no longer be coincident in the physical memory space. The third action points out that the repeated path segment strategy is not as helpful as originally assumed (see section 4.8.3) because it reduces the migration of training data from one path segment the next one via the generalization overlap at the target. This generalization overlap only gets exercised every 5th pass, and so the two neighboring path segments are much more likely to develop different solutions if perturbed by a hashing collision near the target point. Interestingly, the pos-

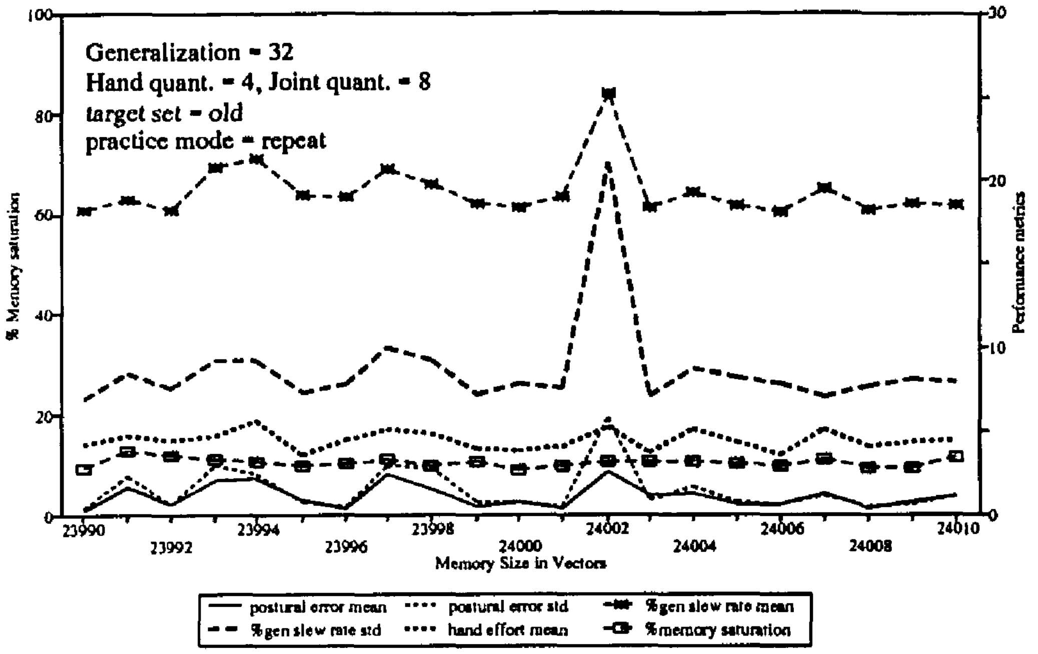


Figure 4.15: Memory Size Dependence for Old Target Set

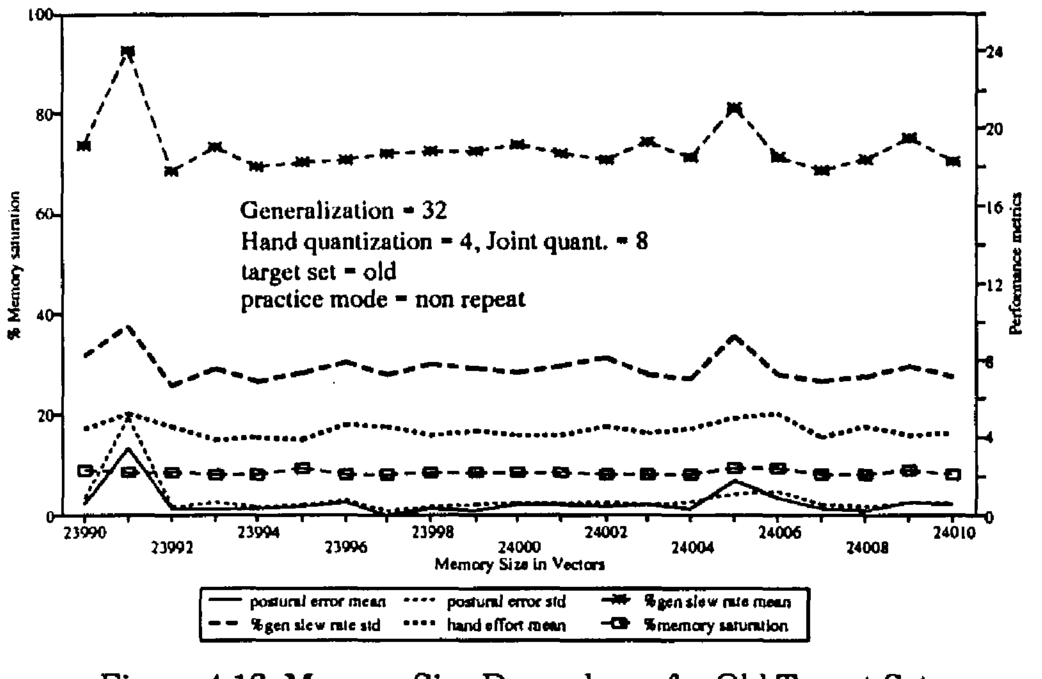


Figure 4.16: Memory Size Dependence for Old Target Set

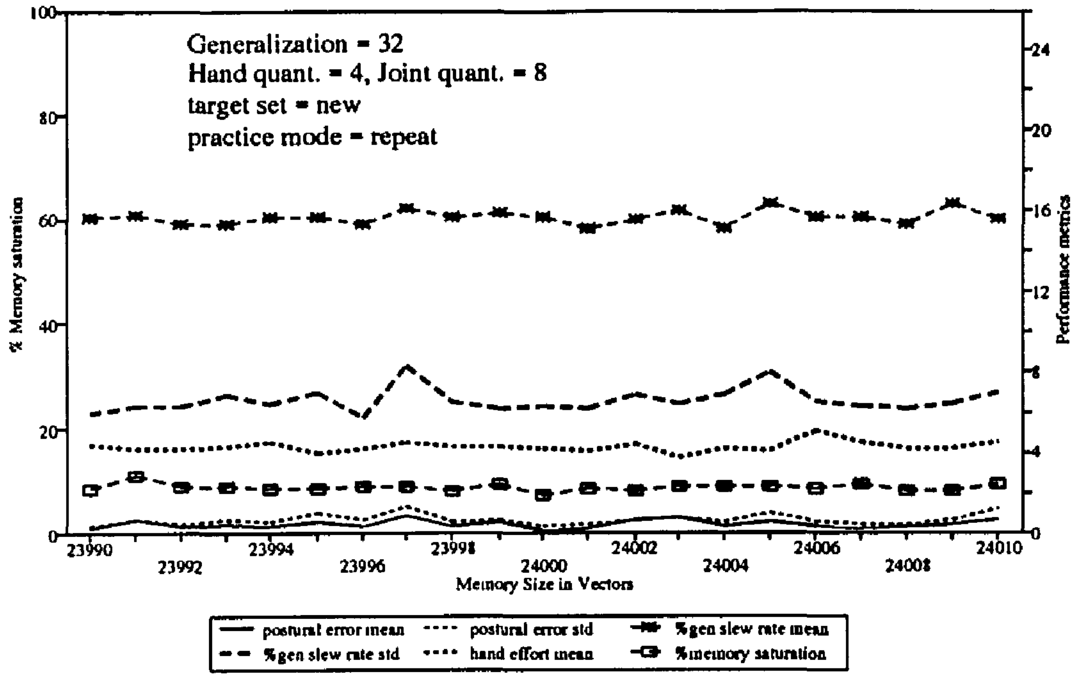


Figure 4.17: Memory Size Dependence for New Target Set

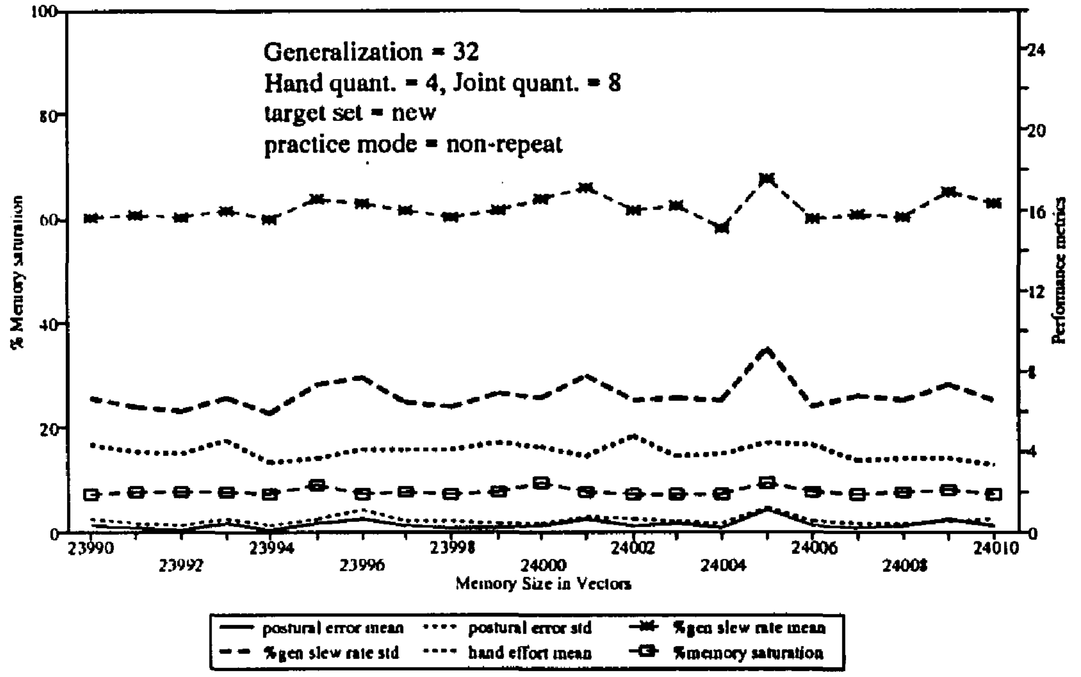


Figure 4.18: Memory Size Dependence for New Target Set

tural difference between these two path segments does converge out after about 3000 epochs.

#### 4.8.9.1 Non-uniform Distribution of Hashing Collisions for High Dimensional Systems.

The previous discussion led to an in-depth consideration of the distribution of hashing collisions in the physical memory. In higher dimensional systems (e.g. 12 or more dimensions) the assumed uniform distribution of hashing collisions becomes gaussian over the virtual address space due to the summation of many samples from uniform distributions, one for each coordinate of the input vector. This consequence of the central limit theorem changes the hashing collision expectations discussed in Appendix B.

This indicates that the amelioration derived from increasing physical memory size is not a uniform effect. The expected number of hashing collisions

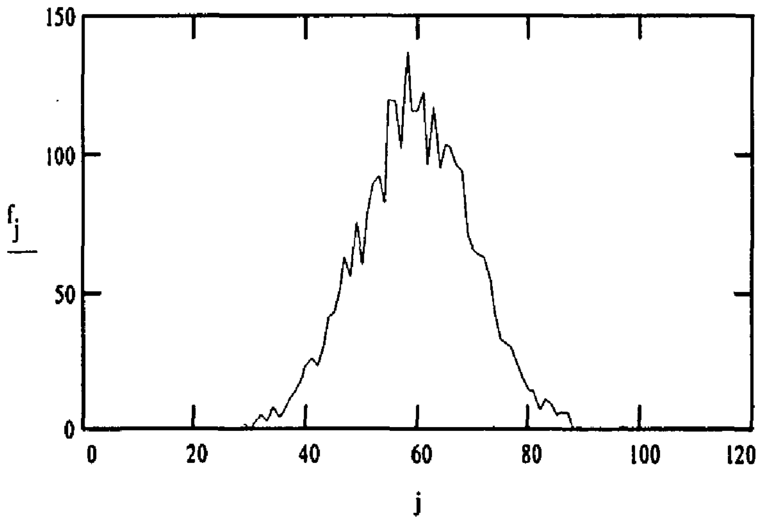


Figure 4.19: Gaussian Distribution of Hashing Collisions

that occur as memory size increases goes down, but as the memory size approaches its theoretical maximum, the fraction of these collisions that are "multiple collision" sites (and thus more damaging) goes up as the distribution becomes gaussian, because in the gaussian case, some weights are more likely to be collision sites than others. This may in part explain why, in the previous section, we found that some physical memory sizes are better than others, even if they are larger. This observation may have implications that deserve further study.

Figure 4.19 is an illustrative example only; it is not directly related to the system under study. It resulted from constructing 12 uniform random sample sets from the interval (0,1) and summing them together generating a sample set ranging over (0,12.00). The plot is a 120 bin histogram of the resultant data.

### *4.8.10* *Memory Capacity*

Here the ultimate capacity of a CMAC is discussed for trajectory storage problems. The notions of data age and bitmapping will be discussed.

***4.8.10.1*** ***Aging of Training Data.*** As trajectory data becomes "stale", i.e. its existence in the CMAC is the result only of training that occurred long ago, without recent reinforcement, it may be that it is there only because it was the result of sweeping out state space regions during convergence of early training, which regions did not become part of the converged solutions, or it may be that it is there to describe how to do things the robot does not do anymore.

In either of these cases, the hashing damage this stale data imposes on more recently trained data becomes innocuous, because it can be trained over with more recent data and its deleterious effect on the system will go away.

In a series of experiments, 3, 4 and 6 jointed arms were trained to repeat a sequence of trajectories with 8 targets. After convergence of each, a set of chained trajectory moves was recorded to a disk file, so the disk file only contained accurate, relevant arm moves. The CMAC was erased and training was resumed using the recorded file to provide the sequence of moves for training. After the recorded file moves were learned, the arm was allowed to train normally on the whole target set until ultimate convergence (4 or 5 passes). The memory saturation was then re-computed. In all trials it was noted that the memory saturation level went down. The decrease was sometimes as high as

| Arm Length | Memory Size | Original Memory Usage invectors | Memory use after retraining | Percent memory Use Reduction |
|---|---|---|---|---|
| 3 joints | 20K vectors | 1501 | 973 | 35 |
| | | | 1013 | 33 |
| | | | 1076 | 28 |
| | | | 1028 | 32 |
| 4 joints | 20K vectors | 1400 | 1149 | 18 |
| | | | 908 | 35 |
| 6 joints | 30K vectors | 2840 | 2698 | 5 |
| | | | 1371 | 50 |
| | | | 1600 | 44 |

Table 4.1: Memory Use Reduced By "Playback" Training

50%. (See Table 4.1.) This indicates that of the state space swept out by the system during trajectory convergence, about 30% to 50% of the state space used was used only during the irrelevant moves encountered in the search involved in early training, and the vectors associated with that phase of training eventually became "stale" and ultimately available for new training instances. The total number of vectors used in initial training in column 3 of table 4.1 is fixed for each arm configuration because the initial training, given the parameters of the experimental setup, is deterministic. The re-training session is not, because there is no way of determining what the "best" training set to record and later play back for training purposes might be. Some of the training sequences learned via this "record and playback mode" were more goal directed than others. The ones that showed the least memory usage after convergence were stable (i.e., they repeated reliably for the same the target set) and so we would expect that these minimal memory usage statistics are sufficient for the problem. These same statistics would be more reliably reflected by a different method of computing memory usage, namely bitmapping.

*4.8.10.2* ***Bitmapping.*** In UNH_CMAC, the level of saturation of a CMAC is computed by simply counting the number of non-zero memory locations in the physical memory. This measure is an approximation of how saturated the CMAC is. It fails to account for *actual* 0 valued vectors; this is a minor effect because zero length move components are infrequent. More importantly, this method treats all "usage" of the memory as equal, even though stale data is irrelevant. An extension of the current UNH_CMAC software should include a bitmapping capability. This would allow each physical memory location to have an associated bit in a bit map. Every time a location is accessed, its bit is set. The bitmap can then be cleared and the system allowed to exercise through a comprehensive set of moves, during which time the bitmap could accumulate new data. Now memory usage can be computed accurately by counting the ones in the bit map. This method would ignore "stale" data.

### *4.8.11 Repetitive and Non-Repetitive Trajectories*

In this section, I describe the results of experiments for trajectory ensembles that are non-repetitive. In this series of experiments, 40 targets were used and the system randomly disturbed each target when it became the active one, resulting in coverage of the workspace more or less uniformly for over 3000 epochs. The result was highly encouraging. Memory usage seemed to level off

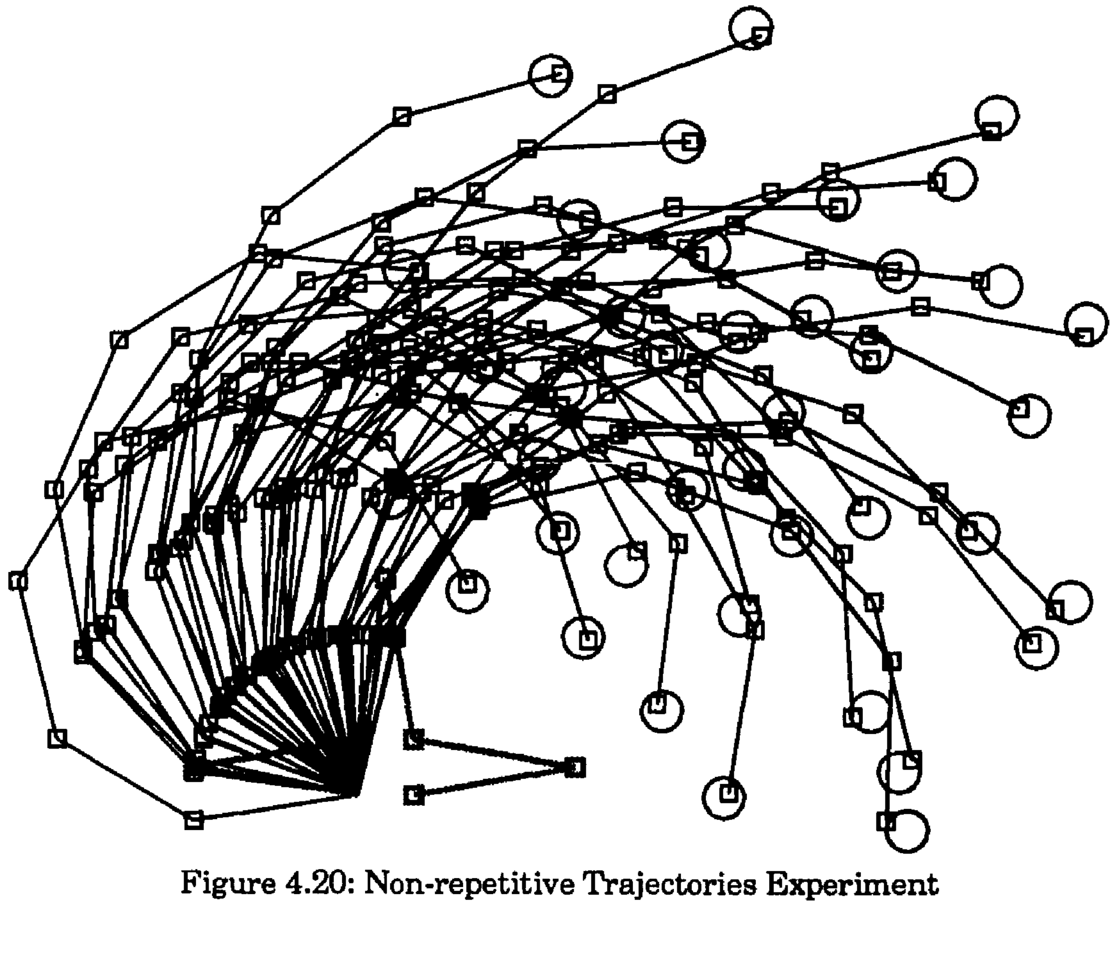


Figure 4.20: Non-repetitive Trajectories Experiment

asymtotically, with little recurrence of heuristic or training critic failures, which is in keeping with the prior observations about large memory capacity.

In addition to the postural animation of figure 4.20, a plot of the memory use and error metrics for the 3000+ epoch long non-repetitive experiment can be seen in figures E.3 through E.5 on pages 214 through 216.

*4.8.11.1* ***Conclusions for Non-Repetitive Trajectories.*** It seems that ARTISTS with constraint satisfaction is extremely robust to new information. After learning a rather sketchy set of trajectories, it continued to execute trajectories that were novel (because the targets were randomly moving throughout the trials) that filled up the workspace quite thoroughly. Once early training was completed, the critics reported nearly zero errors. This is encouraging. It should also be noted that no attempt to re-tune the parameterization for a 6 jointed arm was attempted, other than doubling the generalization parameter.

The fact that the critics found no failures in most of the path segments of these experiments indicates that generalization in the connectionist's sense (i.e. being able to handle novel situations based on prior ones that are similar) was broadly achieved. Some insight into the novel trajectories result can be seen in figure D.8, on page 205. In that figure, little qualitative difference can be seen between the control surface for this novel trajectory and the ones in the other figures of that appendix. Keep in mind that the receptive field for parameterization P=(64,4,8,20000) can span as much as +/- 25° of joint angle around the operating point, if only one joint moves. Because quantization = 8, simulation resolution is 0.1°, and generalization = 64, so $8\times0.1\times 64 = 51.2°$. This means that some influence of training is possible over a range of $+/-51.2\div2 = +/-25.6°$ around the operating point as a result of training at that point, though the effect is diminished at the receptive field extrema due to the linear tapered profile of the receptive field. A full discussion of these issues can be found in An ['91].

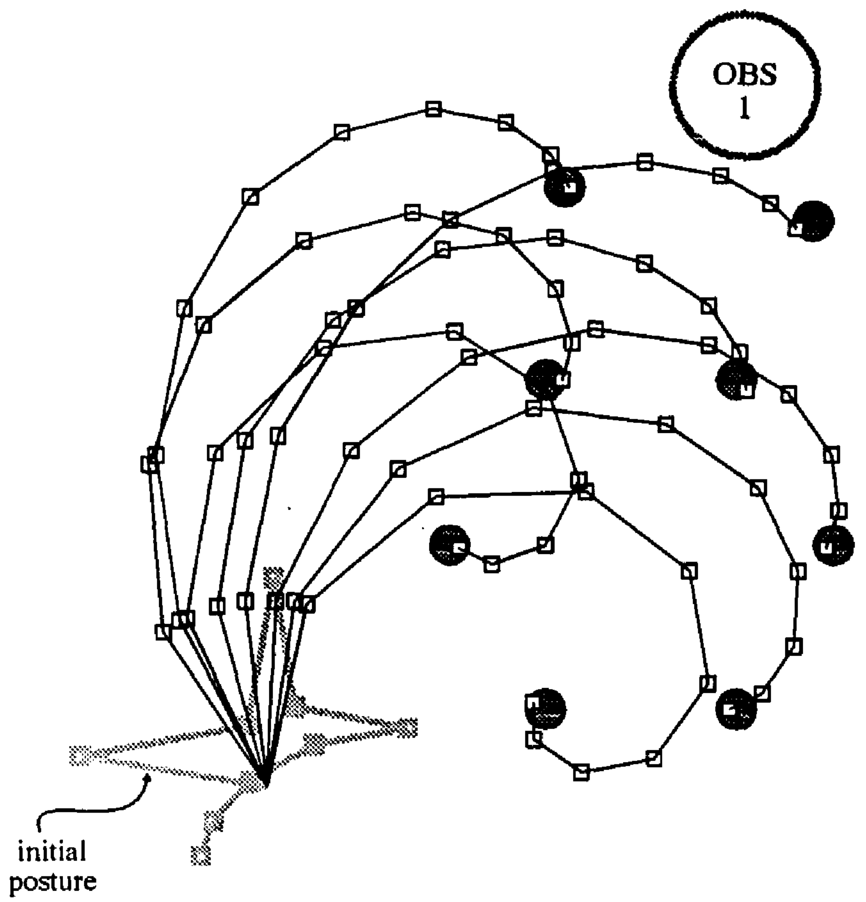
### 4.8.12 <u>Results For Inaccurate Critics</u>

In section 3.2.3, I talked about the critics used in ARTISTS. It should be noted that without the critics, goal directedness falls apart and the redundant system can not learn. The so-called training critic acts as a safety net to prevent inappropriate mappings from hashing collisions in later training from being accepted as valid moves.

In most of the experiments I used the analytical model of the plant as the source of information for both critics. It may be argued that this is not fair. In fact a system able to successfully guess the effect of a move before trying it should be more robust. In an effort to probe this possibility, I devised a series of 3 joint arm experiments using the parameterization, **P**=(64,4,8,20000). For each experiment, I added zero mean white noise to the numerical quantity that both critics used for evaluation. Recall that the critics test a new move to see that it generates a hand move whose orientation is within ( $\varphi_{90}-\varphi_0$) of the desired rectilinear hand space move. ($\varphi_{90}-\varphi_0$) is measured in cosine units. See section 2.4.3. The dot product of the test move and the desired move must be within the range $\cos(\varphi_0)$ to $\cos(\varphi_{90})$. The white noise was crafted in each of 6 1000 path segment experiments to have $\sigma^2$ = 1, 5, 7, 10, 15 and 20% of ($\varphi_{90}-\varphi_0$). For 20% variance noise, the systems proceeded to ask for heuristic help throughout all the experiments. For less, the system's performance over the range of variances was little affected. This indicates a significant degree of robustness in the critic mechanism, so we may find that the system could tolerate a crude forward model, perhaps containing an adaptive forward model estimator. The installation of this new critic mechanism would reduce reliance on the reversible plant requirement discussed in section 3.2.3.1. See section 6.3 for details of the proposed future work in this area.

### *4.8.13* Results For Many-jointed Articulators

Many tests were run for planar arms with 3, 4, 6 and 9 links. The results were all more or less equally successful. In this section I will discuss the trajec-



This sequence of postures results from 150 segments of training of a 9 jointed arm. This means each target to target trajectory was executed 18 times.

Figure 4.21: Target Postures for a 9 Jointed Arm

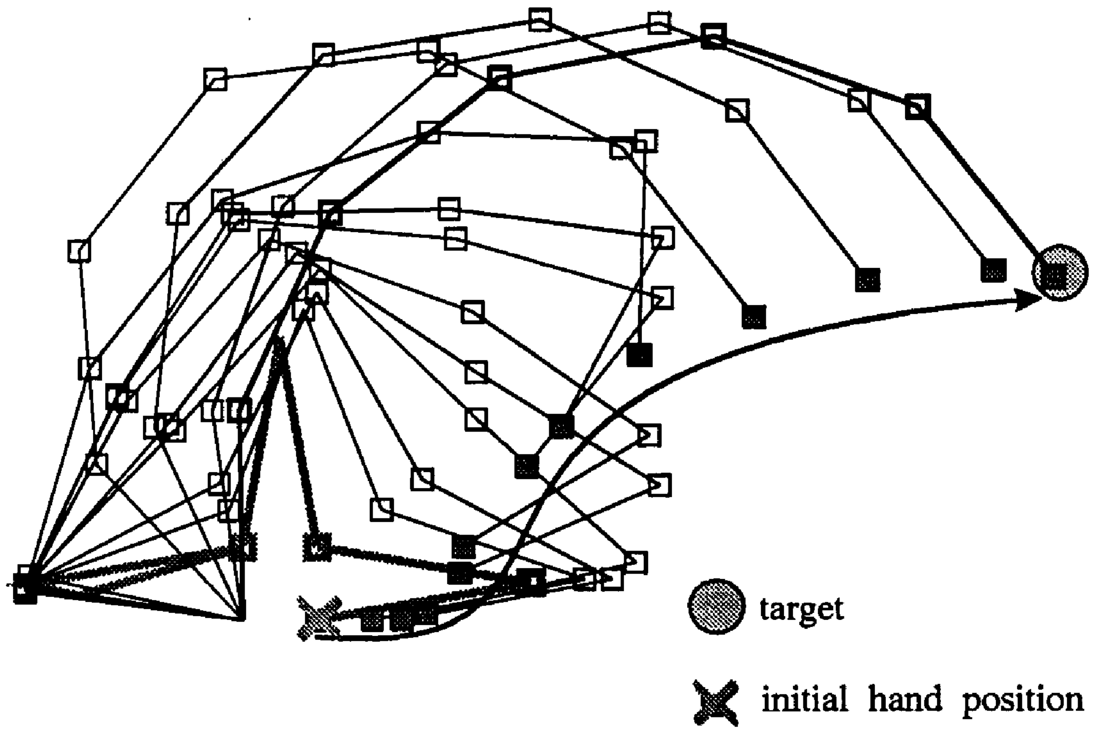


Figure 4.22: The "Uncurling" of a 6 Jointed Arm

tory formations represented by the postural sequence drawings generated in the simulations for 6 and 9 jointed arms.

Figure 4.21 shows the target postures from an experiment in which the initial posture was quite dissimilar from the postural constraint requirements, yet in the first pass, the curvate constraint was achieved, and after 18 passes around the targets, the trajectories showed little or no variation from pass to pass. Figures 4.22 and 4.23 show remarkable and quite robust instances of training 6 and 9 jointed arms (respectively), using equi-angle joint constraints (and joint ratcheting during early training). It is remarkable that the postural constraint mechanism allows the articulator to so successfully "uncurl". Figure 4.23 shows a trajectory that was trained only 5 times. It was one path segment out of an 8 target chained trajectory experiment like the ones in the exhaustive

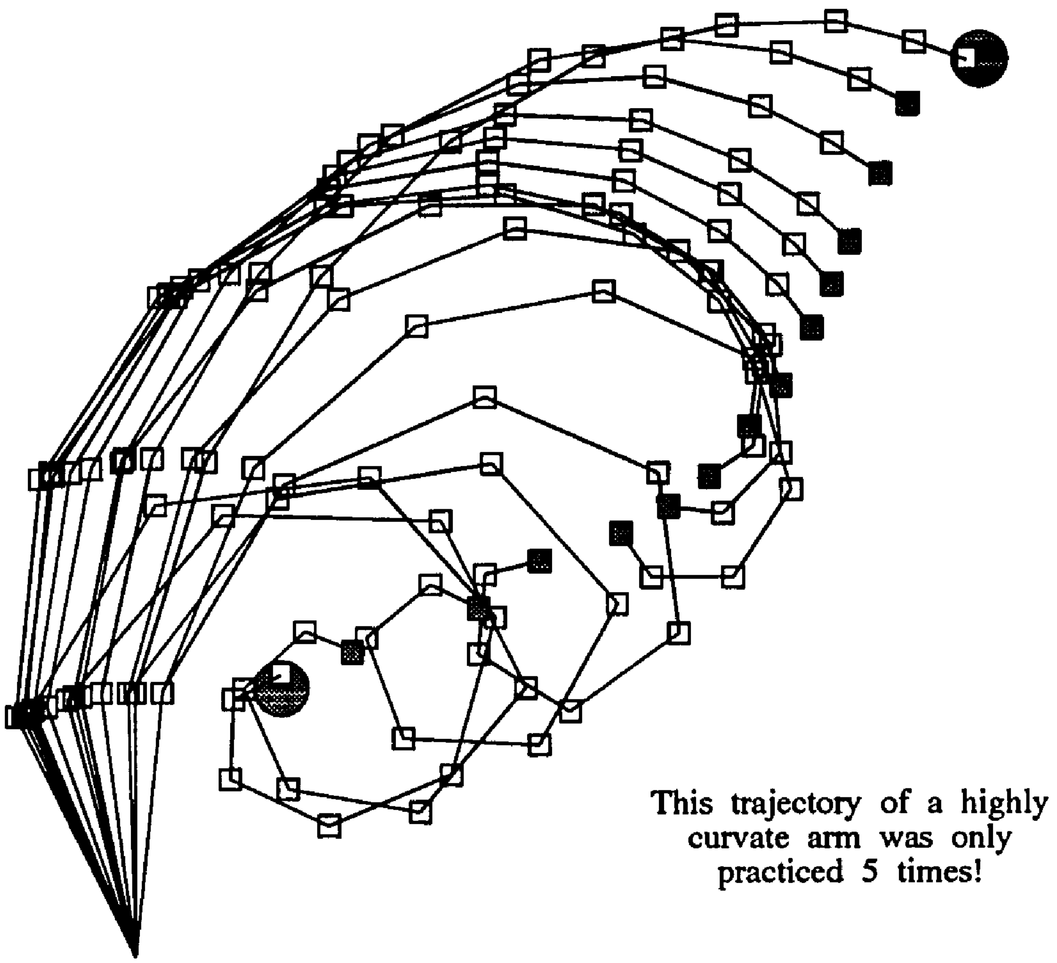

This trajectory of a highly curvate arm was only practiced 5 times!

Figure 4.23: The "Uncurling" of a 9 Jointed Arm

3 jointed arm trajectory experiments used to establish appropriate generalization and quantization.

No attempt was made to fine tune the gross parameterization for 4.23. P=(128,4,8,44) was selected, simply because it seemed that more generalization was required due to the existence of more joints and thus more input coordinates.

*4.8.13.1* ***Conclusions About Many-jointed Articulator Trials.*** This series of experiments was a success. The many jointed articulators learned trajectories in a remarkably short time, and had robust performance that continued to improve over time. Joint stop ratcheting was used in most cases, simply because it reduced memory usage, by cutting down on "stale" data from early training. During early training without ratcheting, the system swept out considerably more state space, but in only a relatively few cases did postural switching occur, resulting in unrecoverable postures. (Postural switching is discussed in section 1.3.1.1 and 4.4.1.1).

## *4.9* ***A Stability Argument***

A rigorous Lyapunov stability proof for the convergence of trajectory formation for the ARTISTS/ARTFORMS system would be useful. This would prove that for any trajectory formed, if the same initial condition and target were presented, then a trajectory very similar to the original would be guaranteed to emerge.

To accomplish this proof, first it would be necessary to neglect hashing collision noise, and then to find a Lyapunov function that emerges from the mathematical definition of the system; the complexities of the CMAC make this difficult. Then we would need to show that this quantity decreases over time as an effect of the execution of the set of equations defining the system. Next, the hashing induced noise must be shown to have a tightly bounded effect on the system, so that the effects of hashing collisions could be reintroduced without destroying the ultimate guarantee of convergence that the proof provides. Such a proof is not within the scope of this work. In fact, for the general case, it may not even be possible. The simple fact that the ARTFORMS critics impose a strict reduction of hand to target distance is a critical starting point for a more informal argument, however.

### *4.9.1 A Lyapunov "Condition"*

A Lyapunov *condition* is *imposed* upon the system, rather than shown to emerge from the system equations. The Lyapunov condition, enforced by the heuristic and training critics says that no move will be allowed that violates the condition that the move must decrease the Lyapunov function, $L(\delta x,\delta y) = \sqrt{(\delta x)^2+(\delta y)^2}$, where $\delta x$ and $\delta y$ are the two coordinates of the hand to target distance. It would be nice if some aspect of the system training guaranteed convergence of this quantity, but that is not the case. In fact, a counter-example for the general case is easily constructed. Consider the articulator in figure 4.24. Joint stops of 90° prevent the target shown in the figure from being reached. Other obstacle or configurational conditions might prevail that could prevent movement towards the target, but which could be circumvented by back-tracking and trying a different approach. Such spurious local and global minimum problems we will term "non-convex obstacle situations" and will explicitly assume can't happen.

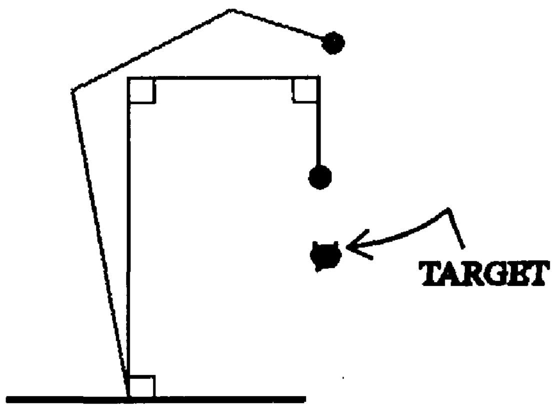


Figure 4.24: A Counter-example for the Lyapunov Proof

That having been said, we now know that reaching the target *is* possible from any initial condition. If that is so, then if we try enough attempts from any given position, we will ultimately find a move that reduces the distance remaining. Such a move will be acted upon and used as a training exemplar, others will be rejected. This disallows any increase of the Lyapunov function (by *fiat*), and since a move that decreases it can always be found, we have therefore assured that the Lyapunov function always decreases.

Thus we now have that the hand will absolutely converge on the target or get stuck somewhere in the attempt, which case is omitted by the exclusion of non-convex obstacle situations assumed above.

If the system were non-redundant, this should finish it, since the hand position is a single valued functional encoding of the joints, and the joints are a finitely valued relational encoding of the hand position. Local generalization favors convergence to single valued encodings of the inverse relation except where the joints are near 180 degrees, in which case alternate solutions are closer together in state space. The sets of alternate solutions allowed by the trigonometric functions (being odd or even periodic functions) are distant one from another in the state space, and thus the system is not subject to postural switching due to non-linear averaging provided that joints are not allowed to be near 180 degrees, in which case alternate solutions are closer together in state space. Empirical evidence shows that for low order arms (three or four joints) this postural switching problem hasn't been a problem for ARTFORMS, so we will also neglect the postural switching effect and assume that for the non-redundant case, the hand position *is* a single-valued functional encoding of the joints.

But ARTFORMS deals with redundant problems, so the constraint equations must enter the argument. The constraint equations are linear, so their least squares expression is quadratic. Thus, as long as the convergence step size is small enough to prevent limit cycling, divergence or chaos, an absolute error minimum in the joint space will be reached by gradient descent. Once that happens, the system model degenerates into a non-redundant one. In other words, the system approaches non-redundancy at the rate the postural error gradient descends.

#### *4.9.1.1* ***A Lyapunov "Argument" and Outline for a Proof.***

The problem remains that the joint space postural convergence can disturb the the progress of the hand towards the target, because the constraint equations are independent of the Lyapunov function. At least four arguments can put this objection to rest. None of these will be argued formally, so the strict Lyapunov proof is still left incomplete. Two of the four require a change in the system. The fourth requires considerable computational consideration, but is promising. The third is a compelling approach and would probably be the best course to persue.

First, heuristic critics could evaluate postural training, and apply a Lyapunov condition to postural training, but that would require a change in the system.

Second, we could show that the hand convergence is robust in the presence of disturbances.

Third, since the postural constraint error will go to a global minimum, it is a decreasing function (though not necessarily monotonic) of time. The disturbance such a function *can* impose on the target convergence is a decreasing quantity. It should then be possible to bound that disturbance by a decreasing "envelope" around the trajectory.

Fourth, the effect that the postural error adjustment imposes on the target convergence is a non-linear function of joint angle changes, so an error term can be constructed that is the sum of the Euclidean hand to target distance and the postural error correction term. Since this is an analytical expression, it may be possible to show that, given the old Lyapunov constraint imposed by the heuristic and training critics, that this is an absolutely convergent function. Failing that, this error term could simply *become* the function for a new Lyapunov "condition" and we would then fall back on the first approach.

There is one final point concerning the Lyapunov stability of ARTFORMS trajectory formation deserving of comment. The input to the system contains a random activation component because random steps are used to search for good moves. Training persists indefinitely, so even if the random component of the heuristic suggestions ceases, random activation remains as a side effect of the hashing collisions in the CMACs, and persistent training will cause that random activation to persist indefinitely. This means that this system can be said to have input with persistent excitation. From a system identification standpoint this is desirable, because it means that the system is not limited to the identification of a system of finite order. Some system designers, in an effort to assure Lyapunov stability of their systems, artificially inject low-level noise just to provide that excitation. Other designers go to great lengths to develop systems that are robust in the Lyapunov sense without persistent excitation, (Lozano-Leal ['89]). This innate persistent excitation in ARTFORMS favors the long term Lyapunov stability of trajectory formation by ARTFORMS.

In the absence of a rigorous stability proof, the evidence of stability is that given a *sufficient gross parameterization* of the CMAC:

- With only sufficient degrees of freedom and no heuristic critic, the system formed consistent trajectories in all observed trials.
- With excess degrees of freedom this breaks down, and postural drift occurs and accumulates causing postures that either stop trajectory formation at a local minimum or reach a condition of nearly sufficient degrees of freedom, at which time trajectories become stably reproducible.
- With proper constraints on the trajectories, a variably specified condition of nearly sufficient degrees of freedom can be imposed.
- All the above holds robustly for memory sizes that are small enough to generate significant hashing noise, and the system can also tolerate noise in its decision rules (the critics) that is substantial.

# *Chapter V*

# *Obstacle Avoidance*

This chapter marks the completion of ARTFORMS-2. In this chapter the development of the idea of a spatially distributed constraint vector field or a spatially distributed non-constant $\boldsymbol{K}$ vector is chronicled. The $\boldsymbol{K}$ vector becomes a non-linear field representation of obstacles encoded as spatially dependent postural constraints. The dataflow diagram of figure 5.1 is similar to the one on page 81 but an additional module is present in the lower left corner, which can adaptively modify the constraint vector that is used in the postural gradient descent computation in the lower right corner of both these figures.

## *5.1* *Preview*

The mechanism for this constraint vector modification is embodied in the the lower left quadrant of figure 5.1. An adaptive memory component (K-CMAC) there models the constraint vector, $\boldsymbol{K}$. It receives input from a unit that is sensitive to the proximity of a joint to an obstacle sensed in a retinal field of view model (R-CMAC). When such a proximity is sensed, the K-CMAC receives a signal that allows it to adjust its model of the $\boldsymbol{K}$ vector to favor a slight change in posture, which in turn is then passed on to the postural gradient computation module. A detailed description of the dataflow involved in this constraint vector modification is found in section 5.3. Before that detailed description ensues, however, we should first try to justify that it is reasonable to expect that we can effectively control the adaptation of postural constraints.

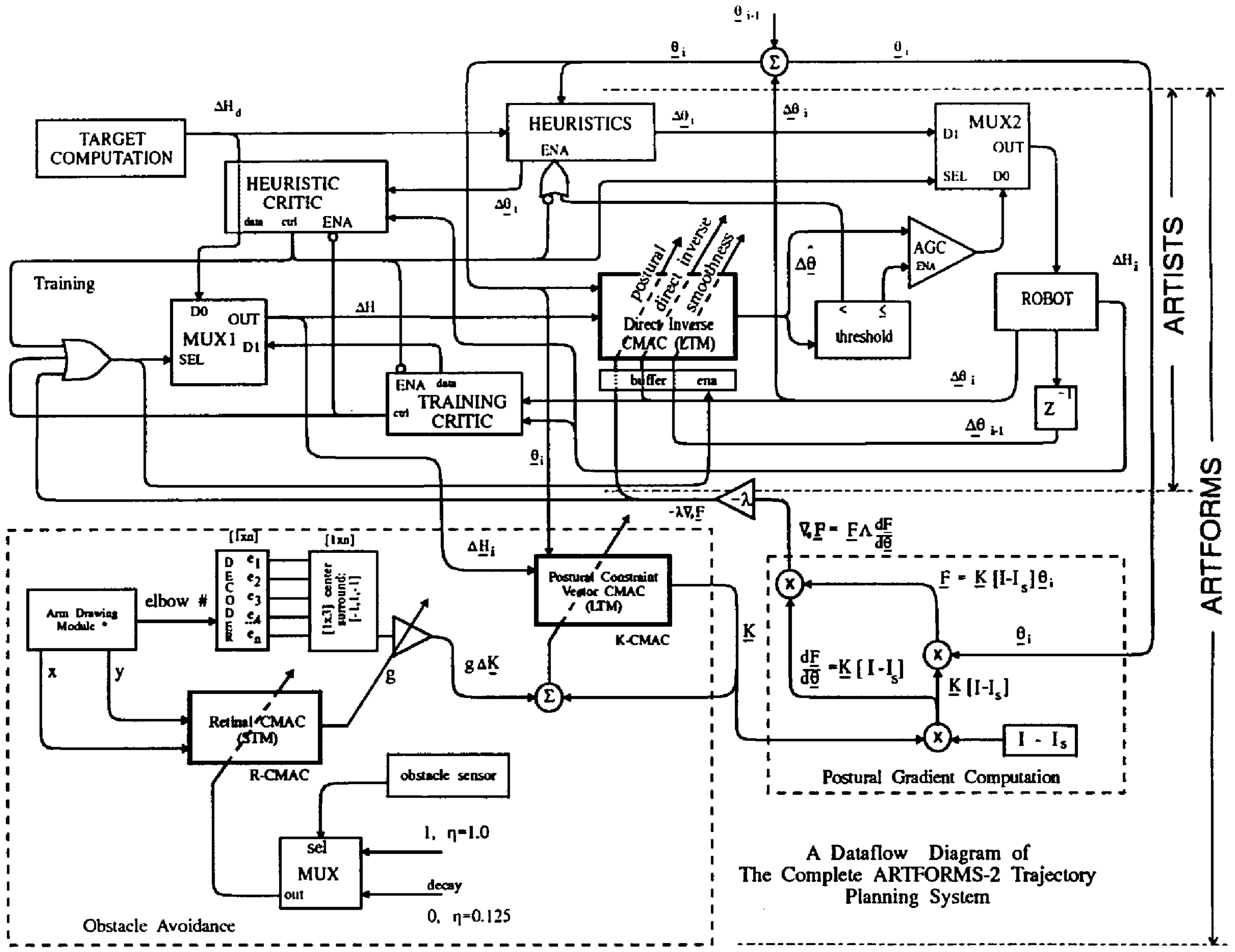
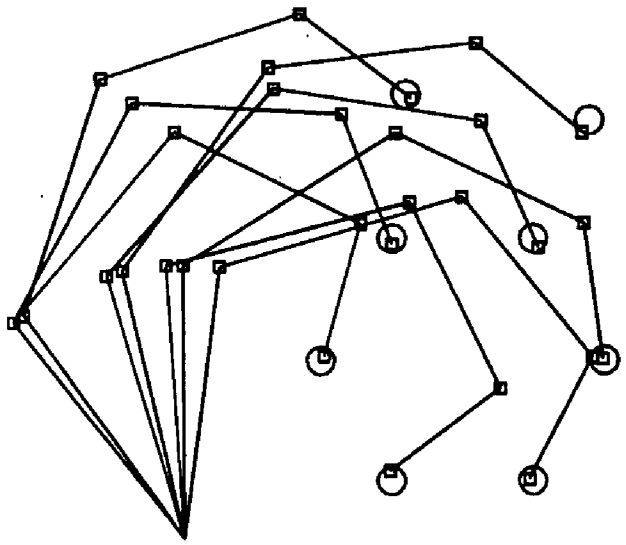
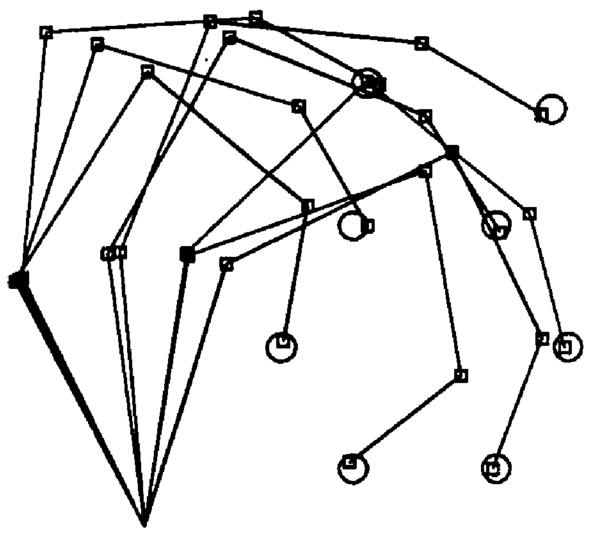

TARGET COMPUTATION
HEURISTICS
ENA
MUX2
D1
OUT
SEL
D0
HEURISTIC CRITIC
data ctrl ENA
Training
MUX1
D0
OUT
D1
SEL
TRAINING CRITIC
ENA data
ctrl
postural
direct inverse
smoothness
Direct Inverse CMAC (LTM)
buffer
ena
AGC
ENA
threshold
ROBOT
$Z^{-1}$
$\Delta H_d$
$\Delta H$
$\Delta H_i$
$\underline{\theta}_{i-1}$
$\underline{\theta}_i$
$\Delta\underline{\theta}_i$
$\Delta\hat{\underline{\theta}}$
$\Delta\underline{\theta}_{i-1}$
$-\lambda\nabla_{\theta}\underline{F}$
$-\lambda$
$\nabla_{\theta}\underline{F} = \underline{F}\Lambda\frac{d\underline{F}}{d\underline{\theta}}$
$\underline{F} = \underline{K}[I-I_s]\underline{\theta}_i$
$\frac{d\underline{F}}{d\underline{\theta}} = \underline{K}[I-I_s]$
$\underline{K}[I-I_s]$
$I - I_s$
$\underline{K}$
Postural Gradient Computation
Postural Constraint Vector CMAC (LTM)
K-CMAC
$g\Delta\underline{K}$
Arm Drawing Module *
elbow #
x
y
DECODER
$e_1$ $e_2$ $e_3$ $e_4$ $e_n$
[1xn]
[1x3] center surround: [-1,1,-1]
g
Retinal CMAC (STM)
R-CMAC
obstacle sensor
sel
MUX
out
1, η=1.0
decay
0, η=0.125
Obstacle Avoidance
A Dataflow Diagram of The Complete ARTFORMS-2 Trajectory Planning System
ARTISTS
ARTFORMS

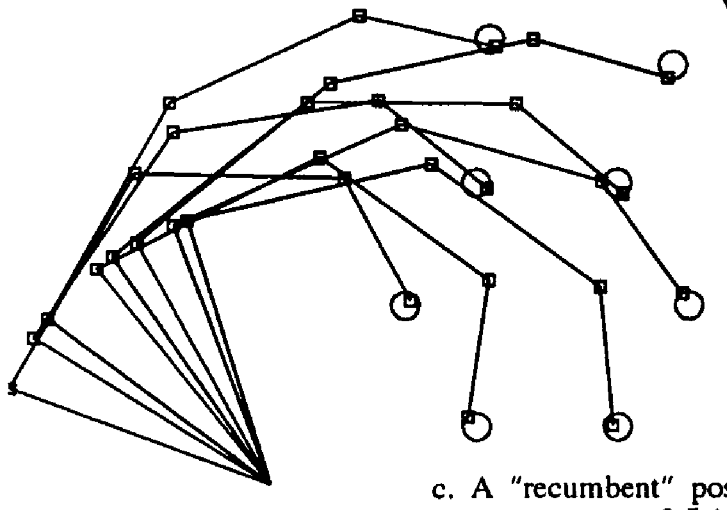
Figure 5.1: Dataflow Diagram of ARTFORMS-2



a. The original uniformly curvate arm for $\underline{K} = [1,1,1,1]^T$



b. A "pronate" posture results from $\underline{K} = [1,1,1.5,1]^T$



c. A "recumbent" posture results from $\underline{K} = [1,0.5,1,1]^T$

Figure 5.2: Postural Changes Due to K Vector Changes

## 5.2 *Modifying Postural Constraints*

The postural constraints that are described extensively in section 4.6 simply place conditions on how the joints are linearly related in a fixed posture. So far this postural relationship is globally applied to the system.

### *5.2.1 Conservation of Memory Usage*

Experiments were performed in which a constant $\underline{\boldsymbol{K}}$ vector was used which defined the linear relationships between pairs of distal joints. A nominal value of (1,1,1,...1) constrains *n-1* joints to be equal. (See figure 5.2a.) The memory saturation level was observed and the vector was changed to a new value, $\underline{\boldsymbol{K}}$ = (1,1,1.5,...1), defining a more "pronate" posture (figure 5.2b), and more training accumulated. The new memory saturation was larger than before. However when the original $\underline{\boldsymbol{K}}$ vector was restored, not only did the arm rapidly resume the old posture, but the level of memory usage remained essentially unchanged,

| 3 Jointed Manipulator | | 4 Jointed Manipulator | |
|---|---|---|---|
| K Vector | Memory Vectors | K Vector | Memory Vectors |
| [1.0,1.0,1.0] | 1389 | [1.0,1.0,1.0,1.0] | 1072 |
| [1.0,1.0,1.2] | 1494 | [1.0,1.0,1.5,1.0] | 1735 |
| [1.0,1.0,1.0] | 1500 | [1.0,1.0,1.0,1.0] | 1755 |
| [1.0,1.2,1.0] | 1608 | [1.0,1.5,1.0,1.0] | 2415 |
| [1.0,1.0,1.0] | 1609 | [1.0,1.0,1.0,1.0] | 2423 |

Table 5.1 Conservation of Memory Use vs. Posture

indicating that no new state space was visited. Next, the $\underline{\boldsymbol{K}}$ vector was disturbed significantly in the opposite sense, $\underline{\boldsymbol{K}}$ = (1,1,0.5,...1), to cause a deflection toward a more recumbent posture (figure 5.2c). Again, more state space was swept out, resulting in a significant increase in percent memory utilization, but again, when the $\underline{\boldsymbol{K}}$ resumed its original curvate value of (1,1,...1), the posture resumed its former state quickly (within 1 path segment) and no significant additional memory usage resulted.

This experiment was performed, with the results shown in table 5.1 for both 3 and 4 jointed arms. The same type of experiment was performed with many arm configurations for 3,4,6 and 9 jointed arms, by *manually* changing the $\underline{\boldsymbol{K}}$ vector during execution of the arm, where the $\underline{\boldsymbol{K}}$ vector was deflected first one way and then another, always returning to a prior condition, and upon return to an old $\underline{\boldsymbol{K}}$ vector, memory usage proved to be a conserved quantity.

### *5.2.2* ***The Robustness of Constraint Training***

The latter results are encouraging. They contain the seeds of the rest of the implementation. Before embarking on that story, I wish to digress and ponder some properties of the constraint trained data.

The postural constraint was met very quickly in every case. Usually postural error dropped to near zero values within a very few path segments. The hand effort error metric, however, persisted in a non-zero state long afterwards. When a new posture was dictated by a new $\underline{\boldsymbol{K}}$ vector, the new posture seemed to develop independently of the hand trajectories in spite of the fact that *training with the postural constraints disturbs the trajectory data.* Furthermore, when two postures were selected that were both previously learned, the switching time from one posture to the other was profoundly short, typically less than 1 path segment.

This evidence is both understandable and perplexing. It is understandable because the postural constraint is direct and strictly goal oriented. In other words, the postural constraint says "change the arm posture", and directly places data in the CMAC that could potentially do this in a single step, but for the value of $\lambda$, whereas the direct inverse training requires feedback for evaluation of success because it changes $\underline{\theta}$ in order to observe a change in $\mathcal{X}(\underline{\theta})$. Recall that $\lambda$ is the step-size parameter or learning rate for constraint satisfaction training. Curiously, however, the constraint training works *in spite of its being required to tolerate this lack of goal-directedness of the direct inverse training.* Another intriguing point is that the value of $\lambda$ was not a critical parameter. $\lambda = 0.05$ and $\lambda = 0.5$ were both robustly tolerated. The former simply resulted in a slightly slower convergence rate in assuming the desired posture. This is at odds with the gradient descent methods used in most non-linear systems. In fact values of $\lambda = 1.0$ were even tolerated (perhaps because the learning rate of the underlying CMAC was set at 0.5). Only when values of $\lambda \geq 3.0$ were used did stability of trajectory formation become an issue.

The original notion (ARTFORMS-1) of obstacle avoidance was never successfully implemented. The reason for that is that it depended too much on serendipity. The old system perturbed trajectories away from the current posture whenever an obstacle was encountered, but no appropriate guidance was provided by the system to give an alternative posture. The result was that when an obstacle occurred, radical changes resulted, and large quantities of state were quickly consumed. An aggravating effect during that stage of development was that the ideal setting for generalization and quantization were not determined yet, and so the system was already operating at a disadvantage.

## 5.3 *Exploiting Postures For Obstacle Avoidance*

All that is necessary for effective shallow search obstacle avoidance is now in place. The essence of the paradigm about to be described is that obstacles will be sensed on a 2 dimensional retina. This then causes a $\boldsymbol{K}$ vector representation to be changed whenever an elbow enters the receptive field of an obstacle feature in the retina. This change will exploit joint pair synergies like those discussed by Hinton ['84].

Refer to the dataflow diagram of figure 5.1 for the following discussion, principally the section labeled "obstacle avoidance" which is enclosed in dashed lines at the lower left of the diagram.

### 5.3.1 *The Retinal STM CMAC*

A CMAC is allocated with 2 input dimensions and one output dimension. The input is computed by the mouse cursor location in the target manager module. Whenever the left and right mouse buttons are depressed simultaneously, a circle with a radius approximately equal to the retinal CMAC's receptive field radius is drawn. That mouse cursor location, in scaled screen coordinates, is used to excite the retinal CMAC. The retinal CMAC (R-CMAC) is then trained with an exemplar value of 1.0 and $\eta$ = 1.0, to allow the feature to be captured rapidly, with a single training instance. The R-CMAC is a short term memory.

## *5.3.2* ***The Spatially Distributed K-vector***

Now, whenever the arm is drawn, each time an elbow is drawn, the elbow coordinates are used to excite the R-CMAC. If an super-threshold value emerges from the R-CMAC, it becomes a scaled sub-unitary gain, *g*, applied to a 3 element center surround vector, [1,-1,1], that is used to adjust the current $\underline{\boldsymbol{K}}$ vector, read from the postural constraint vector CMAC (K-CMAC).

The decoder generates an n-dimensional unit vector with its $j^{th}$ coordinate, $e_j = 1$, and others $e_{i \neq j} = 0$. This vector is then used as a mask to construct the center surround vector, $\mathbf{c}_j$, In the vector $\mathbf{c}_j$, the $j^{th}$ element is -1, the $(j \pm 1)^{st}$ element is 1 and all others are 0. The result of computing $\underline{\boldsymbol{K}}_{t+1} = \underline{\boldsymbol{K}}_t + g \cdot \mathbf{c}_j$ on the system is to cause the $j^{th}$ joint to increase relative to its 2 neighbors for $g>0$, and the decrease for $g<0$.

### *5.3.2.1* ***Practical Considerations Related to the R-CMAC.***

The result of the training of the K-CMAC based on readouts from the R-CMAC is the creation of a spatially distributed $\underline{\boldsymbol{K}}$ vector. There is an effective 2 dimensional to $n+2$ dimensional projection of information from the R-CMAC STM to the K-CMAC LTM.

This STM$\rightarrow$LTM projection is an interesting feature of ARTFORMS-2 and it deserves study in its own right. It is a method of potential applicability for the propagation of constraints from one sub-system to another. We must realize, however, that its presence is an accommodation to the needs of a simulation platform. A real-world physical robot would be better served by a touch sensitive sheath surrounding the arm which would replace the R-CMAC, and provide obstacle avoidance information in the form of directly acquired "elbow proximity" information. The circuitry involved in this could also automatically disable all joint actuators from the disturbed elbow back to the base. This would give the arm some additional compliance, and based on the results in section 4.4.1.2 (page 94) ARTFORMS should automatically handle the disturbance caused by this.

In the simulation, however, the touch sensitive sheath is very difficult to simulate, so the arm is allowed to collide with and even drive through obstacles, relying on the assumption that, in a physical implementation, the touch sensitive circuitry just described would prevent damage[1].

*5.3.2.2* ***Interior and Exterior Obstacles.*** The obstacle avoidance paradigm that is implemented here only considers "exterior" obstacles. These are obstacles that are outside the region enclosed by the combination of the curvate arm and a straight line connecting the actuator to the base. The extension to interior obstacles should be straightforward, but will not be developed in this dissertation, because initial attempts were inconclusive, and time constraints led me to defer this to future work.

## 5.4 Limitations of This Method

The use of direction cosines in the input vector rather than absolute target locations imposes some limitations on ARTFORMS-2. The hand cannot be disturbed from its rectilinear path during a trajectory in order to avoid obstacles. Only the "elbows" can migrate around. With the implementation of absolute target location that may be changed. The result could be trajectories formed like

1 So you, kind reader, should not be troubled by this obvious violation of physics in the simulation!

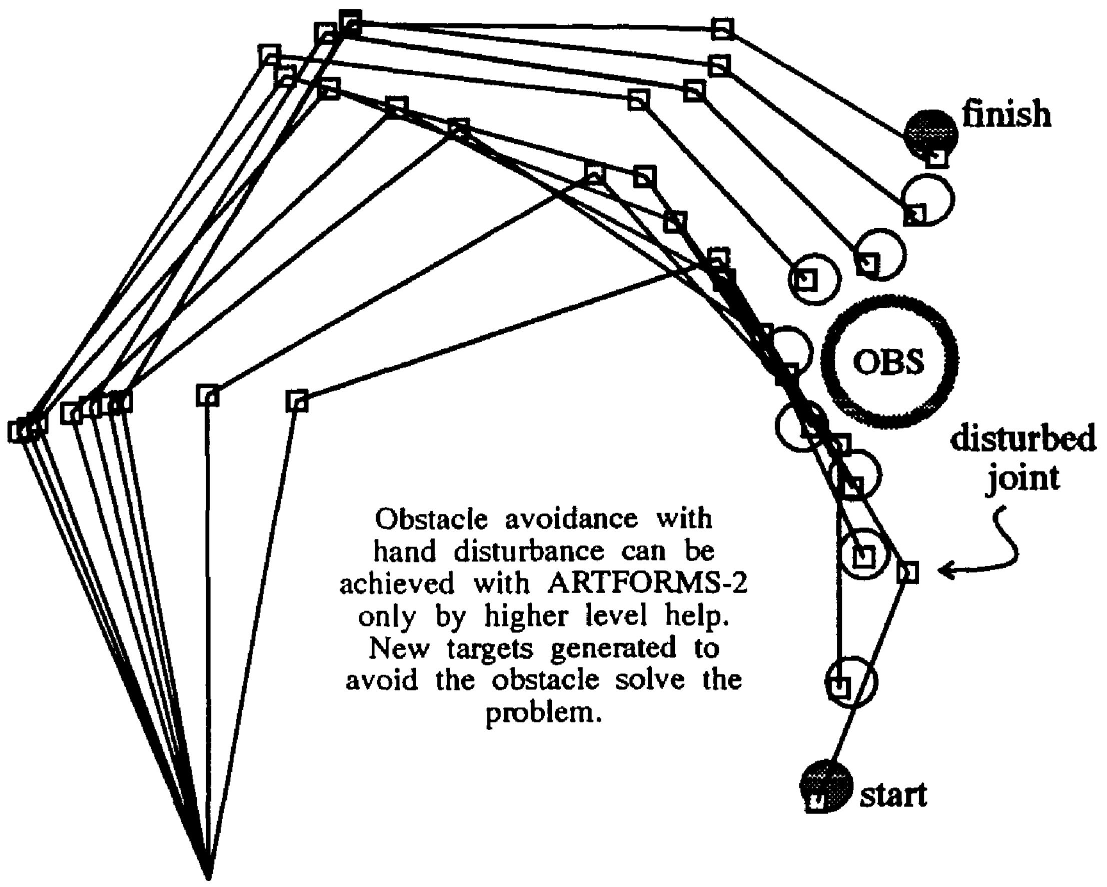

finish
OBS
disturbed joint
start
Obstacle avoidance with hand disturbance can be achieved with ARTFORMS-2 only by higher level help. New targets generated to avoid the obstacle solve the problem.

Figure 5.3 Obstacle Avoidance by Hand Disturbance

figure 5.3. See section 6.3.2 for more discussion of this possibility. The reason for this inability to disturb the hand is that ARTISTS will always train, via the direct inverse training step, nearly rectilinear paths in the hand space. With ARTFORMS-2, such hand disturbance would be handled by inserting new target locations in the workspace. Such a feat would have to be handled by a higher level in the system. Figure 5.3 shows the generation of a trajectory that avoids an obstacle by disturbing the hand. The experiment was successful in adjusting the most distal joint during the first iteration of the path segment to cause the joint to miss the obstacle, but as is shown in the figure, the hand was deflected by the (manual) insertion of several intermediate targets along the trajectory that were placed to avoid the obstacle. Hogan ['92] argues that just such a mechanism goes on in human neuromuscular control in the formation of trajectories, but since the trajectories he suggests are in fact bell-shaped in the workspace, only very few intermediate points are required. If the direction cosine encoding in ARTFORMS-2 could successfully be replaced by endpoint target information as argued above, the hand deflections could be incorporated in ARTFORMS-2. In fact, in the experiments I performed using endpoint data rather than direction cosines, the hand trajectories were usually bell-shaped or sigmoid. The problem with those somewhat ambivalent results was that too much memory use occurred during training, and the critics never went dormant.

## 5.5 Experimental Results

Here I will discuss the experimental results, for various arms, of 4 or more joints.

For these experiments, the lower half of the dataflow diagram of figure 5.1 was implemented and 2 additional CMACs were allocated, one to represent the retina and one to represent the constraint vector field. The R-CMAC and K-CMAC both used C=64, simply because that was the setting used for the main CMAC. I set the memory size of the R-CMAC to be half that of the others (this seemed reasonable due to smaller dimensionality). The R-CMAC used a quantization of 1. It took screen coordinates (1 codon = 1 pixel) so it could reasonably be viewed as a retinal model of the computer's CRT at the resolution of the screen. So on the screen, a receptive field was a square region with a 64 pixel diagonal.

The K-CMAC has the same gross parameterization as the main CMAC, except that its output dimension is *n-1*, rather than *n* (for *n* joints). The reason

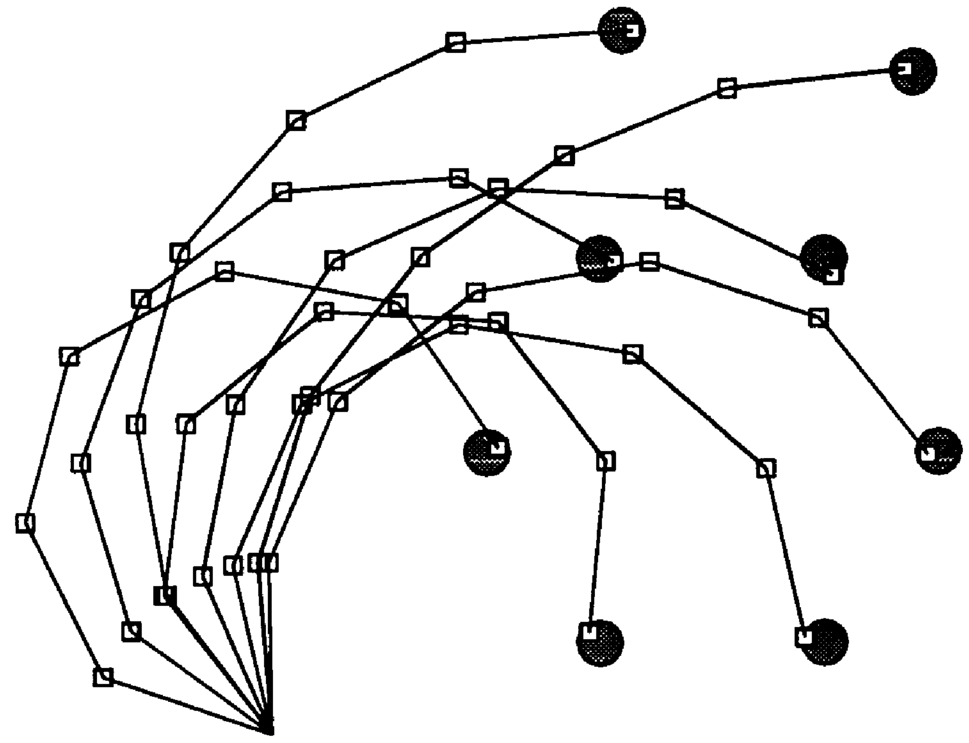


Figure 5.4 A Chained Trajectory Without Obstacles

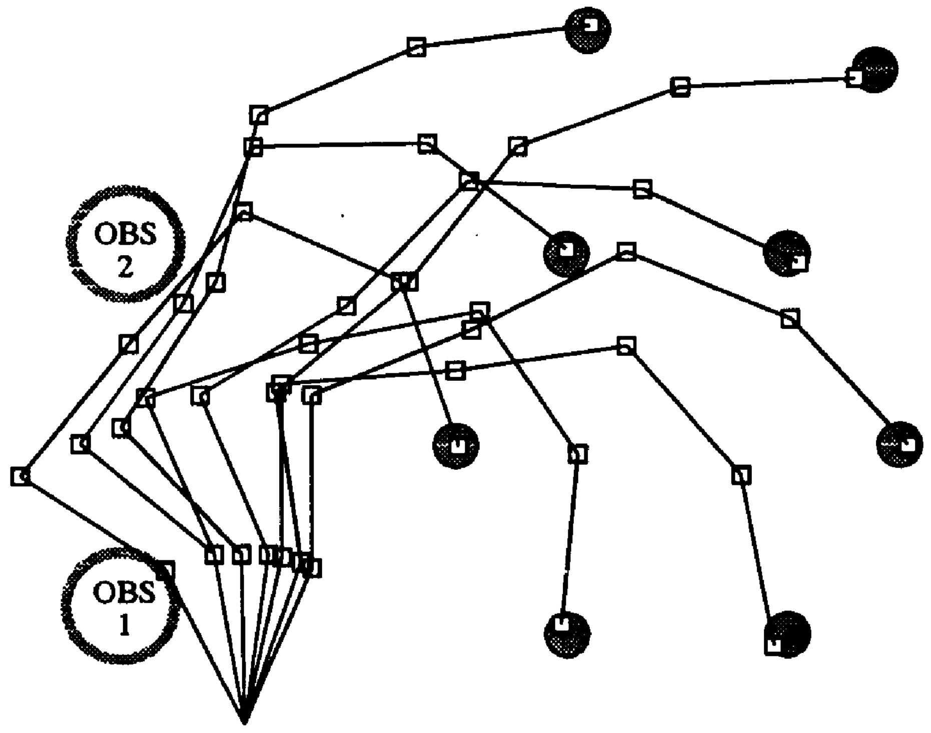


Figure 5.5 A Chained Trajectory With Obstacles

is that the K vector need not affect the base joint since the base angle is a free variable, and thus there is no need to allocate an unused coordinate.

See figure 5.4 and figure 5.5 for before and after results of obstacle avoidance experiments. The results generally were that in one or two presentations of the obstacles, these obstacles were successfully encoded in the K-CMAC. The results shown in the figure are entirely adequate, and the constraint did not seem to propagate into regions of the input space that were not directly related to the obstacle.

The results, in spite of the admittedly haphazard selection of a parameterization, were gratifying. This indicates the system is fairly insensitive to system parameter changes, or I was very lucky (a most unlikely occurrence!).

Some cursory experiments were tried with other numbers of joints, with mixed results. Whenever the length of the arm links were larger than the (approximate) diameter of the receptive fields in the R-CMAC the results were good. In cases like figure 4.21, on page 133, when the obstacle was placed near the distal joints (nearest the hand) the results were poor. The reason for this is that if multiple joints enter the receptive field of the R-CMAC, the (1,-1,1) patterns of the center surround vectors used to adjust the K-CMAC compete and therefore do not give good results. For these cases, in further developments of ARTFORMS-2, a longer center surround vector with a more complex shape that could excite multiple joints near the center and inhibit multiple ones in its periphery is needed. The number of joints to be affected would be a function of the size of the R-CMAC receptive field and the length of the arm links. Of course, as was discussed in section 5.3.2.1, a touch sensitive sheath and a real robot would be even better!

A more comprehensive probing into the robustness of the obstacle avoidance layer of ARTFORMS similar to the level of investigation of the ARTISTS trajectory formation and robustness found in Chapter 4 is beyond the scope of this dissertation.

# CHAPTER VI

# A Framework For Future Development

In the light of the results derived from the simulation described in the previous chapters, it is obvious that some improvements can be made that are rather straightforward changes in the system. It is also important to add some broader conclusions that are compelling and are not directly tied to the experiments. These two areas are discussed in this chapter.

## 6.1 Direct Inverse Modeling is Widely Applicable

Although this project was framed as a robotics problem, it embodies principles of a widely applicable methodology. There exist many applications in industry, science, business and elsewhere in which processes behave in accordance with vector space definitions and in which a sufficient (i.e., adequate, though not necessarily optimal) setting for some parameters must be known in advance to assure the success of the process. Typically, the forward process transform is easy, but the solution of the matrix equations for the inverse transform (which would allow the *a priori* setting of the parameterization) can not be found because of the existence of non-invertible matrices in the system equations due either to singularities or under-determined systems. The adaptive method of learning a representation of an inverse transform may be of wide and profound interest to other industries and disciplines.

## 6.2 Consonance With Traditional Non-linear Approaches

One of the most powerful features of CMAC is its adjustable local generalization capability. This local generalization capability allows systems like ARTFORMS to extend the traditional methods of non-linear analysis around stable operating points. ARTFORMS is consonant with those methods, and allows multidimensional models to be constructed that by their very nature act linearly around any operating point.

The "traditional" method of dealing with a non-linear plant is to develop the differential equations of the plant, determine what are stable operating points of the plant, and then linearize the equations and discuss the approximating properties of the resultant linear system. The assumption is then made that the plant will never leave a state space region around a stable operating point, and so the non-linear nature of the plant's operation far from that operating point becomes moot.

A CMAC, like a computer memory, is a non-linear system. (See section A.1.1.4, page 175.) The CMAC delta rule is, however, a linear operation when viewed at the same operating point each time. Segee ['92] talked about spectral methods of viewing the effect of CMAC training over a range of points, and used that view to explain observed slow learning scenarios for CMACs. In his discussion, he found that the Fourier transform of the profile of the receptive field (linearly tapered in the system under discussion here) explained the slow learning phenomena, because it had spectral nulls at harmonics of 1/C. An ['91] and Carter ['90] found that slow learning occurred whenever the spatial frequency of the exemplar function was a harmonic of 1/C. My experiments performed during the research reported in Carter ['90] indicated that whenever the spatial frequency of the function was at one of these critical frequencies, the read-out of the CMAC representation exhibited aliasing and erratic results with rather high amplitudes, rather than exhibiting an absence of high frequency content. Thus the patch plots in Appendix D, by virtue of having no exhibition of higher spatial frequency content than 1/C indicate that there probably is no such content. This means that, since all the observed patch plots exhibited features that fell within the extent of a single receptive field, then the learning capability of the ART-ISTS CMAC *should* be undiminished throughout the system's operating range, as has been repeatedly observed in practice.

Taking this evidence together, I hazard to suggest, in the absence of a rigorous development, that this CMAC based learning system and ones like it can reliably and stably build models of control surface representations provided that the CMAC meets the spatial frequency constraints just discussed, and the outputs developed cause a generalization slew rate that is small. (See section 4.8.2.6.) If the generalization slew rate is such that a small fraction of the receptive field width is traversed in stepping from one iteration to the next along a trajectory, then we are assured that each incremental training step will *linearly affect* its immediately subsequent neighboring step via the delta rule. Since we can see by inspection of figures D.1 through D.8 that the shape of the control surface function is very nearly linear over sub-patches that are ≈ 10-20% of the extent of a receptive field, this linear averaging is highly appropriate. By the time the operating point has moved sufficiently far along the trajectory that a distinctly non-linear relationship exists between the current "patch" and its antecedent along the trajectory, a new and distinct set of weights are being used than were used by that antecedent. So every point along the trajectory is in a sense operating the delta rule within a linearized region of the overall control surface.

## 6.3 *Adaptive Critics*

The training critic discussed in section 3.2.4 is a fixed law module that critiques the adaptive steps, i.e. steps taken on the advice of the adaptive estimator (the LTM CMAC), to see if it is appropriate. The fixed law it uses is a simple analytical model of the forward mechanics of the articulator. It would be a significant improvement to the system under discussion here to replace this module, as well as the heuristic critic of section 3.2.3 with adaptive critic[1] modules that do not know the exact forward mechanics of the plant.

Section 4.8.12 argued that the system shows sufficient robustness to be able to tolerate such a change in architecture, but for future work, the actual installation of one would be a significant area of study. As an aid to future researchers, a suggested implementation is outlined below.

### 6.3.1 *A Design for an Adaptive Critic*

Figure 6.1 outlines a module that replaces the analytical forward model used by the heuristic and training critics described in chapters 3 and 4. The only difference between the heuristic and training critics in the context of figure 6.1 is that in the former case data from a heuristic source is input, and in the latter, data from the LTM is input to the critic. The critic consists of two CMACs, a forward model CMAC and a confidence CMAC, implemented in parallel data paths

1 Note that this is intended to refer to a "generic" concept of an "adaptive critic" which is distinguished from Paul Werbos's ['90] more restricted definition of an adaptive critic. Here, I am simply denoting a module that is adaptive and whose purpose is to critique actions.

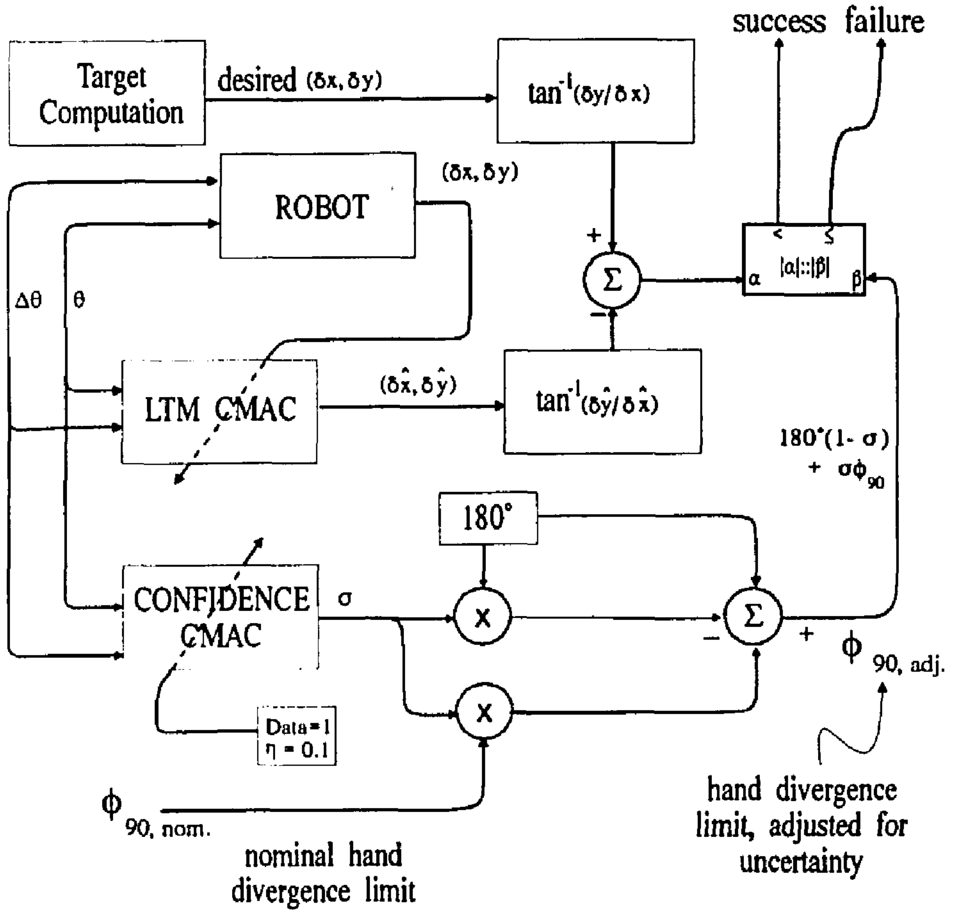


Figure 6.1: An Adaptive Critic Module

and interconnected in a fashion such that the confidence CMAC influences the way the system interprets the output of the forward model CMAC.

Every time the robot is activated, the input to the robot is also applied to the forward model CMAC, and the robot's response to the input is used as the forward model's training exemplar. Take note that this is a single valued, non-redundant transformation, and can thus be learned quickly and reliably by this direct inverse method.

In the ill-fated implementation of Chapter 3, a "distributed plasticity" CMAC was implemented for the purpose of providing a learning rate for another

CMAC with which it was interconnected. In this case, however, the distributed plasticity CMAC becomes a distributed confidence measure for the adaptive forward model. It is a "grain of salt" generator, if you will, that tells the system (as a function of state space location) just how reliable the forward model is.

Recall now how the heuristic criterion works. There is an "allowed cone" angle, $\varphi_{90}$, which defines by how much the hand may diverge from the desired $(\delta x, \delta y)$ target direction vector. If a prediction of the forward model falls outside that cone, the step is deemed a failure. Suppose the critic is very bad. In such a case, the critic may tell us that a move is a failure when in fact it is fine. This is not a happy occurrence, because that move will be rejected as a training exemplar and if many such failures occur, learning will be very slow. If the critic tells us that a move is a success when in fact it is bad, then we may temporarily learn to do the wrong thing. The former case is worse than the latter because, in the latter case, we are at least learning *something* and eventually, as the forward model improves, we will more and more often learn the "right" thing. So the best policy is to be very gullible at first, and as the critic becomes more knowledgeable, be more and more restrictive. The method used to implement this is to expand the allowed cone to encompass any angle if the confidence CMAC reads out a zero value (indicating a naive critic) and use the predefined $\varphi_{90}$ value as the outer limit of the allowed cone if the confidence CMAC reads out a 1.0 (indicative of a near perfect forward model). So there is a new variable value of $\varphi_{90}$, called $\varphi_{90adj}$, which is a linear function of $\sigma$, the output of the confidence CMAC, and has boundary conditions of 180° and $\varphi_{90}$. Solving for the boundary conditions of

$$\varphi_{90adj} = \begin{cases} 180^\circ & \text{if } \sigma = 0 \\ \varphi_{90} & \text{if } \sigma = 1.0 \end{cases}$$

gives rise to the equation

$$\varphi_{90adj} = 180^\circ - \sigma \cdot 180^\circ + \sigma \cdot \varphi_{90}$$

which is the dataflow module in the lower right hand part of figure 6.1. This adjusted angle is compared to the difference between the outputs of the target computation (the desired direction) and the critic, in order to generate a success or failure signal.

Early in the training session, the critics will make lots of mistakes and this will obfuscate things, but in fairly short order these mistakes will decrease in frequency until they eventually go away entirely. In section 4.8.12, we saw that the ARTFORMS-2 system was very tolerant of artificially imposed obfuscatory noise injected into the critic *that never went away.* We can be reasonably assured, then, that the noise the adaptive critic imposes on the system, *which decreases over time,* will be less troublesome than the former noisy critic. The prediction I hazard to make at this point is that once implemented, ARTFORMS-2 with an adaptive forward model critic will perform as well as the ideal ARTFORMS-2 with an analytical forward model critic, but will be completely independent of knowledge of the analytical form of either the forward or inverse models! This, of course, makes the issue of a reversible plant requirement moot. (See section 3.2.3.1.)

### 6.3.2 *Using Workspace Position Rather Than Orientation*

The point has been made by numerous researchers, like Houk ['92], Hogan ['92] and Massone ['89], that target endpoint information is what drives real muscular activation systems. This may mean that ARTISTS can be improved upon by making a modification that replaces the direction cosines in the input vector by absolute coordinates of the target. Such an approach is outlined in section 5.4. ARTISTS was designed to use direction cosines because it originally seemed an obvious, easy, consistent and uniform mapping, and it spanned the necessary space[2]. More discussion of this is found in section 7.2 on page 166.

A cursory attempt at endpoint control produced results that were not conclusive. In general terms, the error metrics did not converge as nicely as for the case of target directed direction cosines, and the critics did not go dormant, they just contributed less and less over time, which is not sufficient. My assumption going into that experiment was that since ARTISTS was tailored to the solution with direction cosines, expectations of a success by simply changing the mapping was overly optimistic. So that too is left as an effort for future work. If it can be accomplished, this would make the extension of ARTFORMS obstacle avoidance into a larger class of problems (consisting of those problems mentioned in section 5.4) complete.

2 It has the same number of degrees of freedom as the arm plus the target position.

# *CHAPTER VII*

# *Conclusions and Future Work*

## *7.1 Conclusions*

These major conclusions were formed in this dissertation:

- Multiple CMACs can be used to model complex systems in a subsumptive fashion. Spatially distributed representations in one can indirectly affect the other. Examples are:
  - spatially distributed plasticity,
  - constraint representation for obstacle avoidance, and
  - spatially distributed confidence measures.

- Direct inverse methods can be used very effectively for redundant systems, given (1) local generalization and (2) on-line training.

- Goal directedness can be forced on direct inverse methods via constraints whose goal is to reduce the dimensionality of the problem. The result is a robust system. This reduction of dimensionality via constraints is probably the root cause of Jordan's forward model successes.

- It is possible that by increasing memory size by as little as 1 vector can result in a measurable change in function approximation accuracy with CMACs, under certain conditions.

- The use of CMACs in control systems can be viewed as consonant with and an extension of the traditional method of non-linear control via linearization around stable operating points

Some more minor conclusions were also made:

- There is a loose linear upper bound on hashing collision damagc for CMACs.
- The summation of CMAC address coordinates that have been uniformly randomized distorts the distribution of the randomized samples from uniform to near Gaussian.

Finally, some slightly speculative results were also asserted:

- A Lyapunov stability argument seems to hold for ARTFORMS-2 trajectory formation, and a rigorous proof may be possible.
- The ARTFORMS-2 system is robust in the presence of noise and this discovery led to a persuasive (though untested) design for a system based on an adaptive critic that will be entirely model independent.

## *7.2* ***Future Work***

Future work for which this dissertation set the stage includes:

- Installing Adaptive Critics: In section 6.3 the notion of installing adaptive critics was discussed. The current implementation is really an idealization of a practical implementation of the ARTFORMS trajectory planner. By removing the analytical forward model, the construction of a fully general adaptive planner for simple planar kinematics is complete.
- Installing Smoothness Constraints: Installation of smoothness constraints with stiffness and/or momentum terms as discussed in section 4.7 should be implemented to determine how effectively they

improve (or thwart) convergence of trajectory formation. This should be especially helpful for long kinematic chains.

- Using Endpoint Control: The minimum dimensionality of an input space must accommodate every important aspect of the input data. In the case of ARTISTS, that dimensionality is $2+n$ for $n$ joints. At any point in the state space, the posture (and thus the hand position) is well defined by the $n$ joints, but the state space input vector must have 2 *additional* coordinates like ($\delta x$, $\delta y$), which provide disambiguation for the case where two trajectories cross in the hand space, and thus have equal joint postures, to define two distinct trajectories in the input space upon which those two points lie, for the case of two distinct targets. If the values of ($\delta x$, $\delta y$) are not equal for these two points, then we may rightly expect that these represent two points upon 2 *distinct* trajectories aimed at *different* targets even though the postures are equal. This seems to infer that the representation spans the necessary vector space of the problem. It is an open question whether absolute target position rather than direction cosines will suffice for that spanning. Clearly, the number of dimensions *is* sufficient.

- Construction of a rigorous Lyapunov stability proof for trajectory formation within the ARTFORMS-2 system.

- Multiple cooperating ARTFORMS-2 systems that have binocular vision input for depth perception, more sophisticated obstacle recognition and can operate multiple cooperating arms in 3 dimensions.

# *APPENDIX A*

# *Future Directions: A Longer Term View*

The next logical step in this research progression is to put ARTFORMS into context, and to broaden the scope of this research to consider a more interdisciplinary approach.

## *A.1 Two Differing Approaches for Interdisciplinary Research*

There are two possible directions a biologically *inspired* motor control paradigm may follow: a philosophical approach and a physiological approach.

The philosophical approach receives deep biological inspiration and insight and proceeds to design on engineering principles which then operate more or less open loop with respect to physiology, once the initial inspiration is complete, This method is exemplified by Albus's ['81] approach.

The physiologist's approach is exemplified by Houk ['90], who persists in adhering to physiological models throughout. The agenda here is clearly first the goal of explication of real neurological systems and only secondarily, engineering systems.

Another approach, the analytical approach, which contains a "symbolic AI approach" as a subset, is a variant of the philosophical approach. This may take a cue from nature, or it may proceed entirely from a mathematical formalism, but it usually attacks a problem at a very high level. The objective becomes to use whatever means are at our disposal to assure as nearly optimal performance with as nearly minimal time complexity as possible. A problem that arises is that these two requirements are strongly at odds with one another. The result is often neither a satisfying application nor a palatable computational budget.

The former two methods are in some ways qualitatively more powerful than the analytical approach because they build on evolutionarily proven systems, without having to reinvent evolution in a bottle, as genetic methods do, which is computationally burdensome. Evolutionary successes can be very instructive engineering parables for the control theorist and engineer. Some adaptive control theorists have argued against emulating nature in complex systems for a number of good reasons. For instance: nature is very inefficient and so blind adherence to models that mimic nature risks inefficiency. Our counter-argument to this class of objections is that there are cases where a system must be designed to be fault tolerant of crude, inaccurate components used in a system in an effort to be frugal, and there are cases where long time delays in systems cause problems that can be ameliorated by "quasi-feedforward" methods that mimic Houk's cerebellar model.

I am not saying that "no engineering solution exists that can not beat evolution at low cost with a lower resolution", nor am I saying that the "evolutionary way" is the only way. It is just simply the case that if a system *must tolerate* crude components (because that is what a cheap design has forced upon it) those inaccurate, noisy, low resolution components pose a problem for the system. Nature has had to handle just those sorts of problems because biological components are inaccurate, noisy and low resolution. One solution (perhaps not the only one) that has worked with a low computational budget (since no computers were involved) is the "quasi-feedforward" solution postulated by Houk. Systems like mine are very similar in strategy, not design, to Houk's, by providing an adaptive feed forward predictive capability in place of tight loop feedback control. Another argument in favor of these kinds solutions is that well designed analytical solutions will not likely be as portable from one implementation to the next unless they are highly adaptive and try, via this adaptivity, to be general purpose solutions, rather than each being specifically tailored to a given application. This latter point suggests that analytical solutions may have problems just from one manufactured unit to the next because of loose tolerances making these different units start to look like different designs.

Finally, the innate time delays of complex systems have deleterious effects as mentioned in section 2.5.2.1, and again, nature has dealt with these problems using solutions like Houk's quasi-feedforward processes.

## *A.1.1* ***Moving Toward More Biologically Inspired Systems***

*A.1.1.1* ***The Philosophical Approach.*** First, we would like to consider some obvious extensions of ARTFORMS under the philosophical approach intended to develop cerebellar methods that can subsume higher level functions.

At the least these would include:

- the fusion of multiple ARTFORMS systems to plan for multiple articulators or other mechanisms.
- the addition of more useful vector representations of articulators' joint spaces, e.g. from vision data extracted from an image processing system that gets input from a binocular vision system, facilitating 3 dimensional postural perception.

But the ultimate target of this method would be to develop systems that resemble Albus's three level hierarchy of the mammalian brain (Albus ['81], p.

184). MacLean ['73] and others have postulated a triune brain hypothesis in which higher levels in the hierarchy exert control on lower ones by inhibition of innate primitive behavior that left alone would proceed in a feed forward fashion (for instance Brooks's['88] walking behaviors).

The hierarchy consists, at the most primitive level, of the *reptilian brain,* with an old *mammalian brain* in the middle, and finally a new mammalian brain occupies the uppermost layer. ARTFORMS now operates at the old mammalian level with incursions into the reptilian level. The Miller/Glanz/Kraft controller might be said to operate at the reptilian level. If your view of reptiles precludes adaptivity, consider Brooks's modified FSMs as an example. MURPHY (Mel ['90]) operates at the new mammalian level.

It is possible to extend spatially distributed parameter systems similar to ARTFORMS well into the new mammalian level. What ARTFORMS is missing is something like a parametric understanding of back-track search. To develop this, some sort of time history capability must be developed.

The incorporation of a more sophisticated Purkinje cell model in the CMAC implementation to conform Houk's (Houk ['89,'90,'91], Sinkjaer ['91]) observation of a hysteresis loop involved in Purkinje deactivation might be a valid approach: delaying the deactivation of receptive fields in CMAC would cause some "smearing" of the receptive fields along trajectories rather than just having a fixed generalization around any state space point as if it existed in static isolation from the dynamics of the system. The result of this time delayed deactivation might be faster learning of trajectories and a better capture of the dynamics of the motion of the mechanism. Such a method of smearing the generalization region would tend to potentiate movement commands when arm velocity is large, which seems appropriate. However, this low level modification may not be necessary, and further, it would only capture past time history, whereas a method of forward prediction is desirable as well.

*A.1.1.2* ***Temporal Lookahead Method.*** Consider figure A.1. In this section, a method of using CMACs to implement a temporal lookahead method

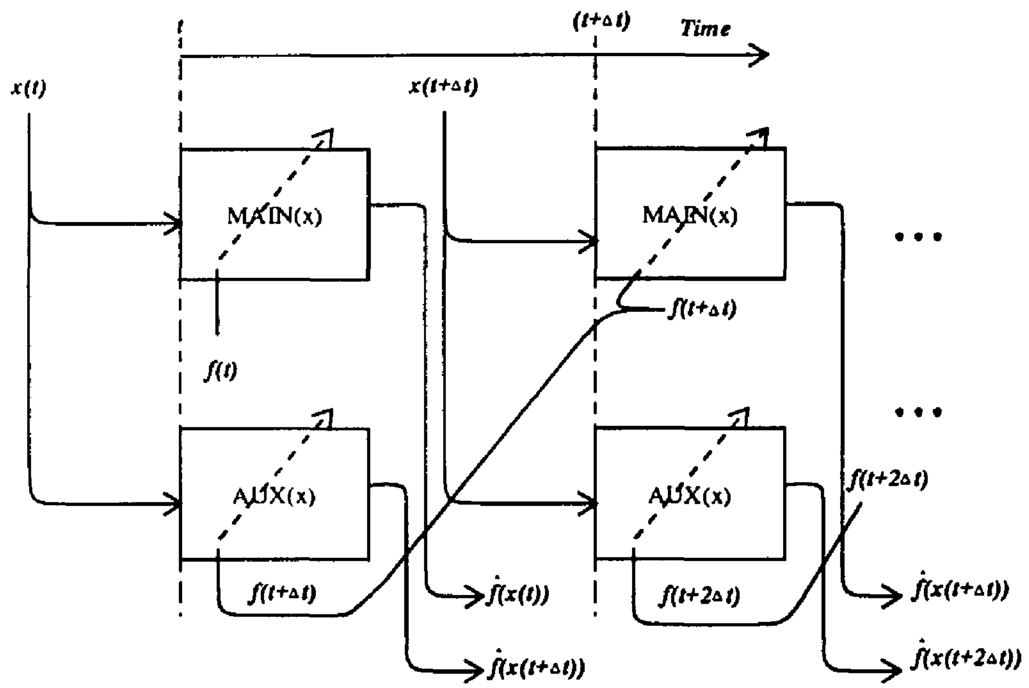


Figure A.1: First Order Temporal Lookahead Method

will be discussed. Temporal lookahead is similar to Sutton's temporal difference method in that multiple networks are used to learn the temporal consequences of actions, i.e., what outcomes result from those actions. To have a single time step lookahead, two CMACs would be required; to have an n step lookahead, *n+1* CMACs are required.

By contrast, Sutton's temporal difference method would train the new information into the same network. The latter method is appropriate for dynamic programming solutions where an action is thrown out based on a reduction in cost of a newly searched path through the given state space point, but the temporal lookahead method does not throw out the prior actions, rather it keeps a complete (spatially distributed) "tree" of consequences active, so that multiple policies can be used to search the same state space. Note in figure A.1 that neither $x$ nor $f$ are *explicit* functions of time, because the CMAC models here model autonomous systems. The AUX CMAC learns the *implicit* time relationship between adjacent steps along the trajectory.

The final suggestion for building on the philosophical approach is to attempt to mimic Houk's ['91] *quasi feed-forward control.* (See section 2.5.2.1 on page 36.) Moderation of predictive and proprioceptively sensed state information can lead to alleviation of problems related to the deficiencies of closed loop feed back control in the presence of obstacles. A feedback system, when thwarted, tends to "push harder" to move its plant under control toward a target. Houk has determined that there is an innate cerebellar/brainstem mechanism that arbitrates the problem of when to push and when to give up. Such an additional feature would strengthen the practicality of ARTFORMS, because currently, ARTFORMS assumes a benign environment: it assumes that the reinforcement signal will arrive in time to prevent system damage, an assumption that is clearly naive from an engineering viewpoint! Section 4.4.1.2 addresses this issue to a limited extent, however.

*A.1.1.3* ***The Physiological Approach.*** Second, (a "first" was in section A.1.1.1!) it would be interesting to apply our methods to Houk's model, rather than moving Houk's ideas into an ARTFORMS venue, in order to follow the physiological approach.

In this proposed research, I suggest that CMACs be used to model the Purkinje cell response in Houk's model. He postulates a positive feed forward ac-

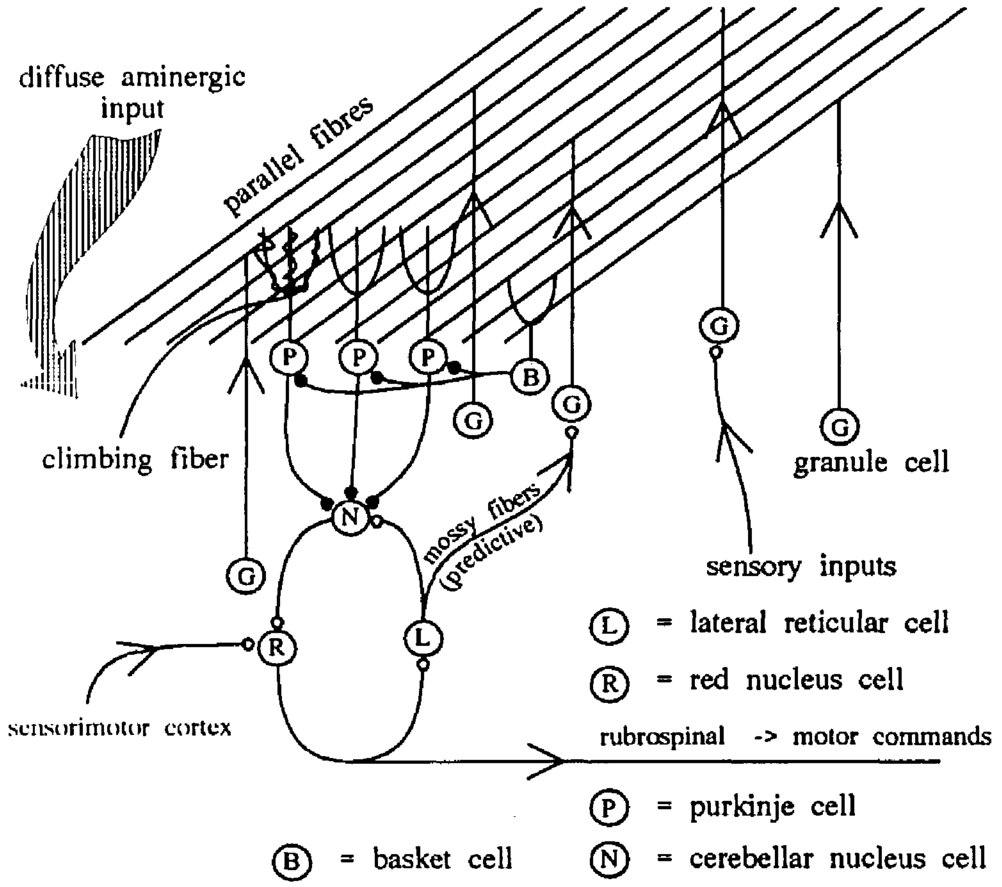


Figure A.2: Cerebellar Adjustable Pattern Generators
(after Houk ['90])

tivation loop comprised of the cerebellar nucleus, red nucleus and lateral reticular nucleus. In his model, motor commands leave the red nucleus and predictive data is fed back into the loop via efferent copy from the lateral reticular nucleus which then merges with proprioceptive input to the Purkinjes. The Purkinjes in turn modulate the loop's output by an *inhibitory projection*[1] onto the cerebellar nucleus, whenever they are strongly selected by the predictive and/or proprio-

1 Note the recurrence of the inhibitory control theme, as in the triune brain hypothesis.

ceptive dataflow via the parallel fibers. See figure A.2. Under the current Houk model, Purkinje cells are trained by climbing fiber inputs, but each weight adjusted by a climbing fiber represents a term in a weighted linear summation. If the projection onto the cerebellar nucleus is also linear (or even a fixed non-linear relationship), the model cannot handle the general case of a neuromuscular-junction/muscle response characteristic that is force-context variably non-linear[2]. If Houk's Purkinjes were replaced by CMACs, leaving the rest of the loop model essentially untouched, a force context variable non-linear response could be modeled. This would at the very least allow for a more realistic arm rotation model, because the assumption of linearity of the muscle force versus $\theta$ that he used was an unrealistic assumption.

One approach for this model would be to allocate one CMAC module for each muscle or muscle group. So each of these CMACs models a population of Purkinje cells that ultimately control the innervation of that muscle or group. This gives rise to the notion of neural modeling by field effect mathematical models rather than discrete neuron models as discussed in the following section.

*A.1.1.4* ***The Field Effect Modeling of Neuron Populations.*** The field of neurobiological research is at a crossroads, not just as relates to motor control, but to memory storage and retrieval issues at large.

It should be obvious from the reading of this dissertation that I consider the "non-linear" approach taken by the connectionists to date as a generalization of the idea of a computer memory. Perhaps we could view this the other way around, in view of the controversy that took place in the late thirties and early

2 In other words, it has a non-linear response that is variable and the nature of the variability is dependent on a spatially distributed force parametric field..

forties between neural networks and Von Neumann advocates. The Von Neumann camp sought to and did mold the direction of 50+ years of computer research and development. The Von Neumann computer architecture could be viewed as a high resolution abstraction of the idea of neural network memory. The effect of this abstraction is that the quality of generalization is abandoned. The result of a non-linear system like a computer memory is that complete isolation of granular concepts is accomplished. A natural neural network can not help but smear information, by virtue of interaction of storage elements. The Von Neumann architecture however is the ultimate non-linear device: there is absolutely *no guaranteed relationship*, linear or otherwise, between the inputs (addresses) and the outputs (data) that remains the same between any one "training pair" and another.

So now, the pendulum swings the other way and connectionists seek to build systems that move away from the "discrete bucket" design of a Von Neumann architecture and toward a "smeared" representation.

It seems in this view obvious that the reasonable approach of building neural models from scratch *without* the Von Neumann model from which to generalize might be the diverse "bottom up" approaches of MLPs, Hopfield nets, Kohonen nets, ART-1,2, or 3, etc. All these methods take a microcosmic view of the network and try to build one by building individual neuron models and connecting them together. This would have been the logical approach if the neural networks camp had prevailed over the Von Neumann camp in the forties.

We are better off in the nineties, however, and it seems retrograde to go all the way back to the McCulloch-Pitts model for anything but conceptual inspiration. The CMAC architecture takes the modern approach: given the Von Neumann architecture, which is thoroughly discretized, how can we build a network that exploits the developments of the last 50 years and yet seeks to overcome the total isolation of stored elements that the Von Neumann architecture imposes on memory. The result is that we build a memory that is non-linear in the same sense that a Von Neumann memory is non-linear, i.e., it is a computer memory of sorts. In local areas, however, a relationship *is* imposed on the contents of the memory that may be linear or it may be non-linear, but it is *consistent and applies to all the state space that the memory defines.*

What we define in such a system is a field model of a memory. Consider the definition of a field in Marshall ['87]:

> "A *field* is defined as the mathematical specification, in terms of position variables and time, of a physical quantity, such as temperature, in a given region."

This is an intuitively satisfying definition of a field that is easier to resolve with the current argument than the more rigorous one found in many mathematics texts like Adler ['67] as an extension of the mathematical concept of a group, which is in turn an extension of a ring. It can be seen that a CMAC as viewed in figure 3.5 fits the definition. In fact, the iterative method of solving LaPlace's equation using nearest neighbor averaging bears a lot of similarity to the CMAC method. The difference between the two methods is that the CMAC method attempts to build a field representation of whatever is presented to it and the NNA algorithm can simply be shown (Noble ['67]) to approximate as closely as desired any field that obeys LaPlace's equation. In both these methods, a local field value is asserted at each mesh point, and the conglomerate effect of many such training instances is that this local field effect propagates outward from each training instance until its effect becomes negligible.

One might well ask: What this has to do with neurophysiology?

Simply this: Neurons have local effects. They also have spreading global effects as they innervate target populations, and as neighboring ones interconnect. It should be possible to build field models of the global effects of these neural projections, based on specific postulated microcosmic hypotheses about neural function. It will be much more feasible, given the current state of the art in computation to try and build models, like CMACs, that can "virtualize" the concept of individual neurons in field effect models of neuron populations and to study the net effect of the superposition of these field effect models. The macrocosmic observation of these net synergistic effects may be a much more effective way to vindicate particular microcosmic hypotheses than the discrete implementations of the microcosmic mechanisms in a computer model that (again, given the current state of the art) is not implementable, or whose implementation is unsatisfying in its simplicity.

The suggestion is then that 2 approaches be taken:

- 1. Build macroscopic field effect models of postulated microscopic structures within the neuron (like dopamine receptors, for instance) and observe the effect of these fields taken in superposition with other postulated models, e.g. initiation of motion in a Parkinson's disease model or the degree of tremor induced in a Huntington's disease model.
- 2. Build macroscopic field effect models of postulated CNS structures. For instance, if we built a field effect model of the cerebellar cortex, and connected it to a loop of modules modeling the field effects of the red nucleus, cerebellar nucleus and the lateral reticular nucleus we could observe the macroscopic behavior of these modules taken in cooperation in

an architecture that connects them together in ways that we postulate occur within the CNS. We could then observe the motor control capabilities of such interconnected modular systems. We would then build each module through architectural decomposition as in approach 1.

Why might these approaches work? Consider the field of electromagnetics. Since Maxwell's introduction of his equations describing electromagnetic fields in the 1860's, we have enjoyed more than a century of success in studying electromagnetism, guided largely by these simple equations, which are founded on models of atomic particles that in view of today's knowledge of quantum mechanics are incredibly naive, perhaps as much so as McCulloch-Pitts is a naive model of the neuron.

Another case history is Miller ['78, '78a]. He described cardiac function using a field effect model based on volume electrical current density. This model was based on differential elements which were naive models of electrical activity in cardiac muscle cells. This approach predicted activation waves across the cardiac muscle and EKG readout that matched clinical observations very accurately.

Francis Crick ['89] argued that we need to concentrate on every detail of actual neurons without wasting our time building networks of simple naive models like perceptrons[3]. Maybe this argument should not be thrown out entirely, any more than it should have been argued during the early 20th century that the quantum physicists should cease and desist because we had not yet

3 In all fairness, his principal argument was against the back propagation algorithm, and there, his argument is hard to refute!

enumerated all the implications of Maxwell's equations. If they had, we might not have gotten around to understanding sub-atomic mechanics until well into the 21st century. But similarly, we should not concentrate exclusively on the microscopic, bottom up approach, because if we wait to view complex neural systems behavior until the microscopic knowledge is fully enumerated and until computer science has evolved to the point of being capable of modeling large scale systems based on the results of said research, none of us alive today will ever see a truly neural-like complex system under study!

In summary, I advise a dual approach: two camps should cooperate, one elaborating on microscopic mechanisms, the other on field effect macroscopic models. If postulations of one camp can be converted into implementations of features in the other's models, we might see development in the field of neurophysiology paralleling that in physics. In physics, engineers continued to develop elaborate communications and transportation systems based on Maxwell's equations while the quantum physicists amended and generalized these equations based on sub-atomic hypotheses and experimental vindications in the lab. Many of these amendments have found their ways into practical engineering systems. If we can emulate the physics research model, someday we may hope to be knocking at the gates of a universal field theory of neural activity, much as today's physicists still hope to do in the place where field theorists and quantum mechanics theorists get together.

## APPENDIX B: An analysis of the probability of hashing collisions.

The probability of ANY collision between 2 random inputs is:

for $m := 500, 600 .. 20000$ ,  $Pa(M,C) := 1 - \left[\frac{(M-C)}{M}\right]^C$

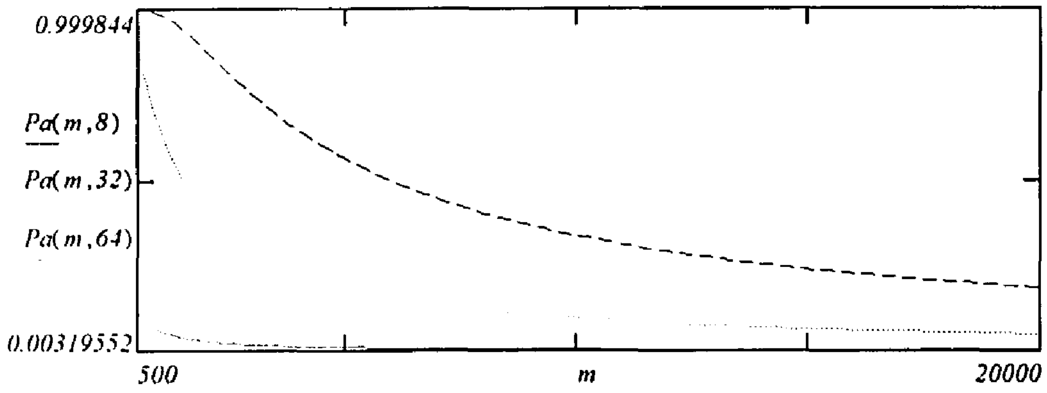


Figure B.1: The Probability of *Any* Collisions Between 2 Random Inputs

Now, given C is the generalization parameter, or the number of memory cells in a CMAC receptive field. M is the total number of memory cells, then

$\left(\frac{C}{M}\right)^N$ is the probability of picking of C of those elements (with replacement) from a C sized subset of M things. $\left(\frac{M-C}{M}\right)^{C-N}$ is the probability of the other C-N choices, where they are NOT in the C sized subset, but are elsewhere in the M sized set.

Finally, since there are $\left(\frac{C!}{N!\cdot(C-N)!}\right)$ ways to pick all committees of N elements out C things, we have the Probability of exactly N collisions between 2 random inputs

$$Pn(M,C,N) := \left(\frac{C}{M}\right)^N \cdot \left[\frac{(M-C)}{M}\right]^{C-N} \cdot \left(\frac{C!}{N!\cdot(C-N)!}\right)$$

Figure B.2 shows a plot of this quantity.

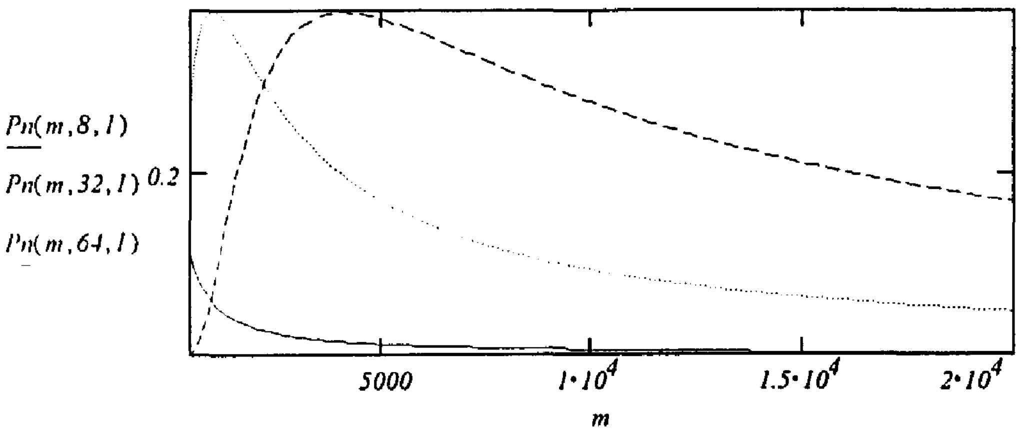


Figure B.2: The Probability of *Exactly* N Collisions Between 2 Random Inputs

The probability of *ANY* collision between random input and existing training is similar to the above, but internally, the role of C in the absisscas above is taken by the quantity: $u = U/M$, where $U$ = the number of vectors used in the current model, and M = the total physical memory vectors available.

$$Ec(m,u,C) := 1 - (1 - u)^C$$

$$u := 0,.01..4$$

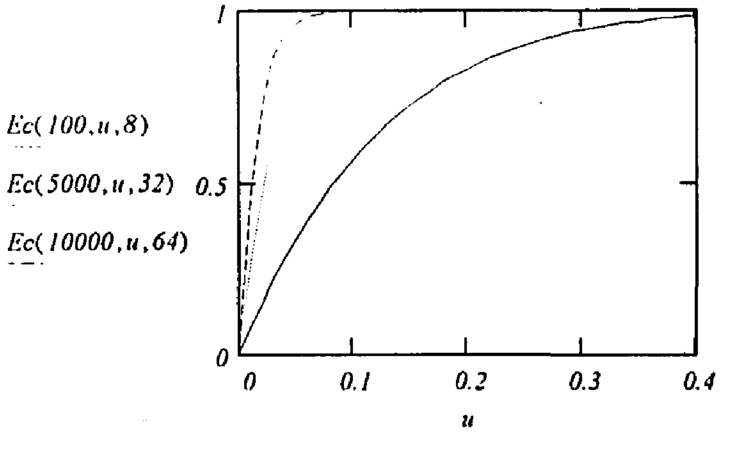


Figure B.3: Probability of Collisions For Any New Random Input

There is a dependence on generalization, but not on physical memory size. This resonates with our experience that has shown us that whenever memory saturation exceeds approximately 30%, performance degrades because hashing collisions become prevalent.

For systems like ARTISTS and ARTFORMS, this 30% upper limit on memory is not very worrisome. There are 2 factors which ameliorate the problem:

1. Only "active" memory usage is important. Any state space that was visited during the process of converging to a solution but subsequently was not revisited during the execution of the solution trajectories will simply be overwritten by new experiential data. This means that hashing collisions resulting from old training data gradually will be attenuated by new training just as old coherent training will. The degree to which one wishes to preserve old training data in the face of new novel data is controlled by adjusting the memory size to be large enough to fit the complexity of the trajectory ensemble expected. In the body of the thesis an experiment was performed to demonstrate how the memory usage drops off significantly after training of trajectories has converged.

2. The ARTISTS architecture includes an adaptive critic that evaluates a proposed move. If the move is not appropriate, a heuristic is used to generate an appropriate approximate move which, upon being trained into the system, will cause the inappropriate move to decay away via re-training.

Concerning both these two points, it is noted that the system has already been shown to be robust with respect to memory size. The following analysis will further substantiate a claim that we need not worry about hashing collisions, given that memory size is set for a "reasonable" value. The rule of thumb to use here is just that common sense prevail: if possible use 20,000 vectors rather than < 10,000 vectors, if the memory is available. If not, just be aware that potential problems may arise.

For the remainder of these examples, u and C will be fixed. C will be set to a value that will be similar to a nominal value used in the arm experiments, and u will range from "naive" to fully saturated memory.

$$C \equiv 32 \qquad\qquad u := 0, .01 .. 1$$

The probability of exactly N collisions between random input and existing training is a binomial distribution:

$$Pc(u,N) := (u)^{N} \cdot (1-u)^{C-N} \cdot \frac{(C!)}{(N! \cdot (C-N)!)}$$

which is analogous to the function *Pn(M,C,N)* derived above.

We'll compute a few of these to see what they look like:

$PC(u) := Pc(u,3)$ $PC1(u) := Pc(u,6)$ $PC2(u) := Pc(u,16)$

$PC3(u) := Pc(u,24)$ $PC4(u) := Pc(u,30)$ $PC5(u) := Pc(u,32)$

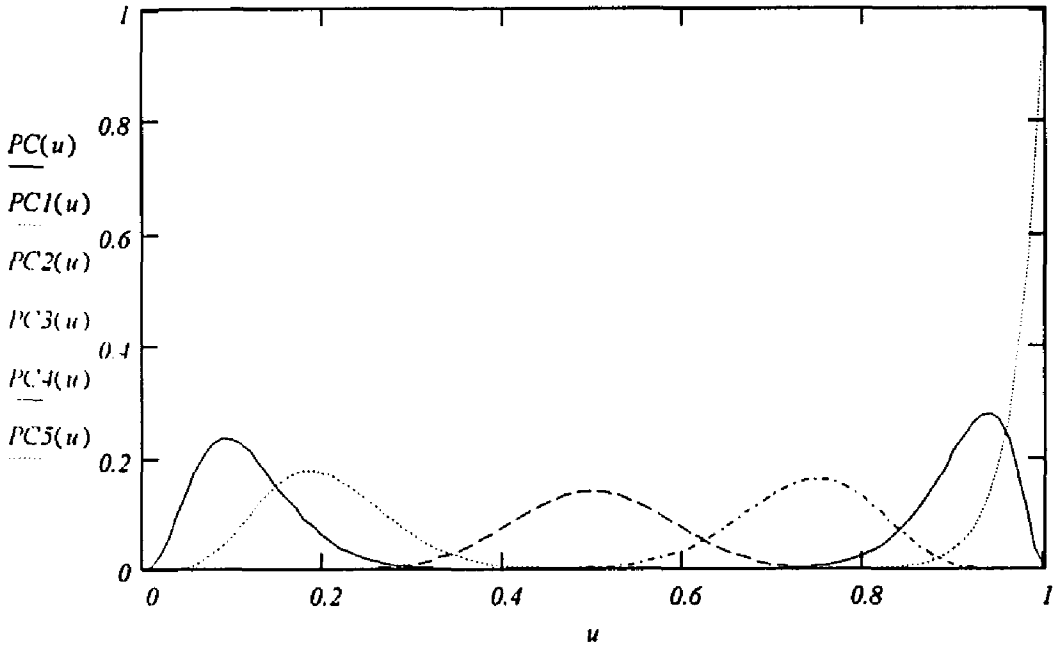


Figure. B.4: Probabilities of N Collisions for New Random Inputs (and existing old data)

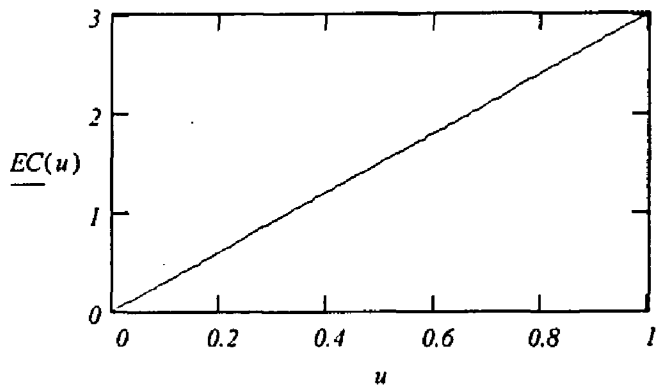
Finally, the expected number of collisions between existing training and random input is just the sum of $Pc(u) \times N$ for all N from 1 to C.

$C := 3$

$n := 1..C$

$$EC(u) := \sum_{n} u^{n} \cdot (1-u)^{(C-n)} \cdot \frac{C!}{n! \cdot (C-n)!} \cdot n$$

It's to be expected that the binomial "humps" on the right side of the plot of *EC* will be amplified and the ones on the left will be relatively attenuated in the multiplication by N, so we'd expect the sum to be a monotone increasing function of some form, but surprisingly, this expected value function turns out to be just the linear quantity, *uC*, as seen in figure B.5.



Figur. B.5: Expected Value of Number of Collisions for a New Random Input

This means that we can expect performance to degrade *at worst* linearly with increasing memory usage, (or decreasing memory size). By *"at worst"*, I mean that the expectation plotted above holds for random inputs, which on average are unlikely to overlap due to valid generalization interaction. However, if the inputs are from an ensemble of training instances that are highly correlated, (as in an ensemble that forms a spatial trajectory), there is a strong expectation that most of the state space visited by the trajectory is the result of multiple overlap of receptive fields. This means that happenstantial collisions will will be better tolerated than if the inputs were all intended to be parts of discrete state space regions as might be the case with, for instance, a pattern classification problem with many distinct classes and few members in each class.

The probability expressions above are from unpublished results associated with [Miller, '90b, '90c]. These expressions were used with random input data to validate the hardware CMAC design and vindicate the equivalence between the hardware and software designs. The result that the expected number of collisions is a linear relationship in *m* is a new result.

The only proof of the new result I've been able to construct is rather messy. A summary of the proof follows. The insight required is to note that in the summand,

$$u^n \cdot (1-u)^{(C-n)} \cdot \frac{C!}{n! \cdot (C-n)!} \cdot n \quad ,$$

$\frac{C!}{n! \cdot (C-n)!} \cdot n$ is simply the product of $n$ and the binomial coefficient, $\binom{c}{n} \cdot n$.

In the sum, only the leading term, $Cu$, which occurs when when $n=1$ survives. It is the only term that contains $u^1$. The other terms are annihilated by adding in expansions of $(1-u)^{C-n}$ that follow, which have alternating signs and leading coefficients that are also binomial coefficients. A more detailed treatment is on the next few pages.

This is a proof of the assertion that

$$E_C(u\,,c) = \sum_{n=1}^{C} (1-u)^{C-n} \cdot \binom{C}{n} \cdot u^n \cdot n = uC$$

| C-n | Each row here are coefficients of the binomial expansion of $(1-u)^{C-n}$ | | | | | | | Each column is a sequence of $n \cdot \binom{C}{n} \cdot u^n$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | C=4 | C=5 | C=6 | C=7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | $4u^4$ | $5u^5$ | $6u^6$ | $7u^7$ |
| 1 | 1 | -1u | 0 | 0 | 0 | 0 | 0 | $12u^3$ | $20u^4$ | 30u5 | $42u^6$ |
| 2 | 1 | -2u | $+u^2$ | 0 | 0 | 0 | 0 | $12u^2$ | $30u^3$ | $60u^4$ | $105u^5$ |
| 3 | 1 | -3u | $+3u^2$ | -u3 | 0 | 0 | 0 | 4u | $20u^2$ | $60u^3$ | $140u^4$ |
| 4 | 1 | -4u | $+6u^2$ | $-4u^3$ | $+u^4$ | 0 | 0 | 0 | 5u | $30u^2$ | $105u^3$ |
| 5 | 1 | -5u | $+10u^2$ | $-10u^3$ | $+5u^4$ | $-u^5$ | 0 | 0 | 0 | 6u | $42u^2$ |
| 6 | 1 | -6u | $+15u^2$ | $-20u^3$ | $+15u^4$ | $-6u^5$ | $+u^6$ | 0 | 0 | 0 | 7u |

Table B.1: A rearrangement of the summation of terms for the expected value of number of collisions

The inspiration for this proof comes from the observation that the expansion of $(1-n)^{C-n}$ has as its expansion a binomial series with alternating signs.

In the table above, the columns in the right hand half of the table represent the values of the quantity $n\cdot\binom{C}{n}\cdot u^n$, taken with $n=1$ at the tail of the arrow. In the table, each column vector, $V$, from bottom to top, starting at the head of the arrow for that value of C. So defining $V^4 \equiv V\Big|_{C=4}$, observe that the column vector $V^4 = [\,4u \;\; 12u^2 \;\; 12u^3 \;\; 4u^4\,]^T$. The sub-diagonals should also be read from bottom to top. For any C, ignore all rows for $(C-n) \geq C$, as these rows would represent negative values of $n$.

In the expression of $E_c$, it is clear that each term of $(1-n)^{C-n}$ must multiply with the component, $v_n$, and this is true for all $n$.

So if we form dot products, $D_i^T \cdot V$ with the n subdiagonals of the lower triangular binomial expansion matrix on the left, starting with the subdiagonal at the head of the corresponding arrow and finishing with the main diagonal, we can observe that the sum $\sum_{i=1}^{C} D_i^T \cdot V$ of these $C$ dot products, for a given value of $C$, is a slight rearrangement of the series, $E_c$, for that value of $C$. Only the first term of the first dot product survives, because the alternating signs along the other subdiagonals cause the terms of each of those dot products to cancel each other out. After that cancellation, all that is left of the series then, for any value of $C$ is $uC$.

Example: $E_c = V^4\begin{bmatrix}1\\0\\0\\0\end{bmatrix} + V^4\begin{bmatrix}-3u\\1\\0\\0\end{bmatrix} + V^4\begin{bmatrix}3u^2\\-2u\\1\\0\end{bmatrix} + V^4\begin{bmatrix}-u^3\\u^2\\-u\\1\end{bmatrix}$. Expanding each term for the value, $V^4$, given above this 4 term series becomes:

$$= \quad uC \; + \; 0 \quad + \; 0 \quad + \; 0$$

It should become apparent after trying this construct for all values of $C \in \{4..7\}$ and for $n \in \{1..C\}$, that the pattern persists regardless of the size of C.

This is not a satisfying proof of the conjecture, $E_C(u\,,c) = \sum_{n=1}^{C} (1-u)^{C-n} \cdot \binom{C}{n} \cdot u^n \cdot n = uC$, but the search for a concise inductive proof has been in vain so far, and this will just have to do!

# *APPENDIX C*

## *A Study of the Linearity of Inverse Differential Kinematics*

This appendix describes some analysis that predicts a generalization slew rate of ~11% for parameterization $\mathbf{P}_4$ and of ~20% for parameterization $\mathbf{P}_1$, which is in agreement with experimental results. It also illustrates that inverse differential kinematics is linear in a rather broad sense. This is of course not true for inverse kinematics.

I will make some simplifying assumptions for the case of a redundant arm with 3 joints to allow plots to be drawn of the kinematic transform. I assume that the differential joint steps will be equal for all three joints. This reduces the problem to one with a closed form solution. This is also reasonable in view of the equi-angle constraint training of the ARTISTS model.

So as not to have to resort to linearization, the inverse kinematics is not solved. Rather, the forward kinematics is shown in figures C.1a and C.1b for various values of $\theta$.

Now the differential kinematics is shown for varying step sizes in the traces of figures C.1c and C.1d. The pertinent point to notice in these plots is that the plots seem to be linear scaled versions of one another, for values up to .1 radians. As the increment increases the plots start to separate in phase, but note that $5^{*}dK(0.01) \approx dk(0.05)$ appears to hold as closely as the eye can distinguish.

## *C.1* A Discussion of Generalization Slew Rate.

The parameterization chosen in section 4.8.2 of $\mathbf{P}_4$ = (64,4,8,20K) performed well. What physical interpretation can be attached to this, and why was it a reasonable choice? The answers to these questions will aid us in the future in assigning a parameterization *a priori*.

In the simulation space, the joint angles were discretized such that 1 discrete simulation unit (su) = 0.1°. A quantization of 8, (qnt_vec = $[,,8,8,...]^T$), means that one codon unit change requires a change of 8 su. So 1 codon unit (cu) = 8x0.1° = 0.8°.

In the simulation space, the direction cosines are encoded such that the vector $[100,0]^T = \hat{i}$, a unit length vector in the $x$ direction. If the codon is set by qnt_vec = $[4,4,...]^T$, then 1 cu subtends a hand displacement of 4/100 = 0.04 linear units.

Now in each coordinate, a receptive field will subtend C codon units. So two postures will generalize if only one joint changes by an amount of

$$\theta_i^j - \theta_i^{j+1} < C \cdot 1cu \le 64 \cdot 0.8° \le 51°$$

or if only one hand space coordinate changes, by an amount

$$h_i^j - h_i^{j+1} \le 64 \cdot 0.04 \le 2.56 \ \textit{linear units.}$$

Of course this is absurd, because the coordinates of the input vector are all coupled by the mechanical linkage, so no one component is at liberty to vary independently. Since all steps are nominally 1.0 linear unit in the hand space, then all hand moves will be on the order of

$$h_i^j - h_i^{j+1} \approx 1.0 \textit{ linear units, (lu).}$$

In fact, the average vector in hand space should be $[\sqrt{1/2}, \sqrt{1/2}\ ]^T$.

$$h_i^j - h_i^{j+1} \approx 0.707 \textit{ linear units, (lu),}$$

so an average hand move will cause a receptive field change (generalization slew rate) of $n{\cdot}0.04 = 0.707$, so n $\approx$ 0.04/0.707 $\approx$ 18 cu, *per coordinate*. Meanwhile, as the hand moves, so move the joints. To cause generalization to occur between succesive postures along the trajectory, no more that 64 - (2x18) = 28 cu can be contributed, in toto by the $n$ joints. Thus, if for 3 joints, the joints each move $\frac{28}{3} \cdot 0.8° = 7.47°$ then no generalization will occur.

How many steps will a receptive field span? The forward kinematics can be expressed, in general as

$$\mathcal{K}(\theta) = \sum_{i=1}^{n} l_i cos( \sum_{j=1}^{i} \theta_i ) ( -1)^{i-1} \hat{\imath} + \sum_{i=1}^{n} l_i sin( \sum_{j=1}^{i} \theta_i ) ( -1)^{i-1} \hat{\jmath}$$

where $l_i$ are the lengths of the links, $\theta_i$ are the n joint angles, and $\hat{\imath}, \hat{\jmath}$ are the unit length basis vectors of the hand space.

For 3 joints, links of 15 lu each, and equi-angular constraints, this becomes

$$x = K_1(\theta) = l_1 cos(\theta_1) - l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1 + \theta_2 + \theta_3) \textit{ and}$$

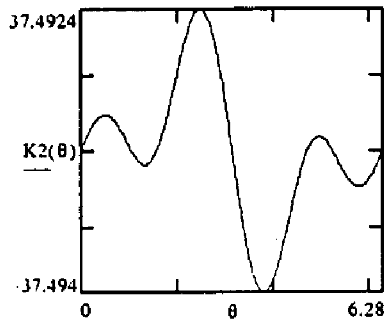
$$y = K_2(\theta) = l_1 sin(\theta_1) - l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1 + \theta_2 + \theta_3)$$

These can be simplified using sum of angle formulae to derive the functions K1 and K2 of figure C.1, (equations C.1 and C.2).

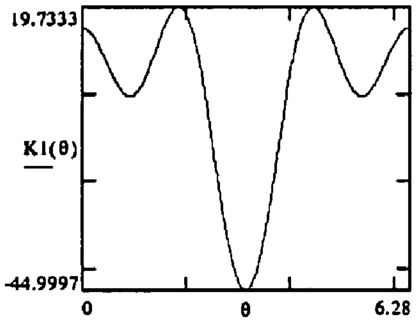
$\theta := 0, .01 .. 2\cdot\pi$

$K2(\theta) := 15\cdot((\sin(\theta) - \sin(2\cdot\theta)) + \sin(3\cdot\theta))$

$K1(\theta) := 15\cdot((\cos(\theta) - \cos(2\cdot\theta)) + \cos(3\cdot\theta))$



a. The y value of K(θ)



b. The x value of K(θ)

$K2(\theta) := -30\cdot\sin(\theta)\cdot\cos(\theta) + 60\cdot\sin(\theta)\cdot\cos(\theta)^2$

$K2(\theta) := 30\cdot\sin(\theta)\cdot\cos(\theta)\cdot(-1 + 2\cdot\cos(\theta))$ (C.1)

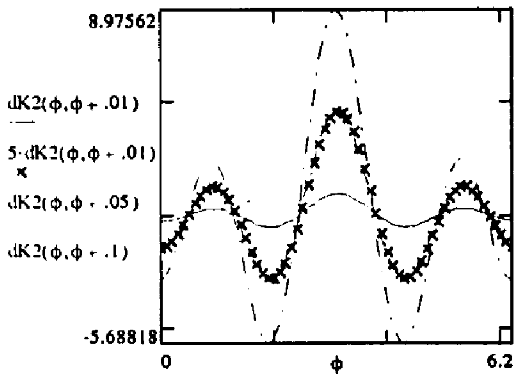
$dK2(\theta 1, \theta 2) := K2(\theta 1) - K2(\theta 2)$ (C.3)

$K1(\theta) := -30\cdot\cos(\theta) - 30\cdot\cos(\theta)^2 + 15 + 60\cdot\cos(\theta)^3$

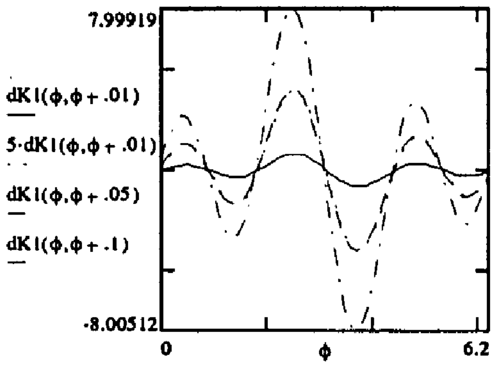
$K1(\theta) := 15\cdot(-1 + 2\cdot\cos(\theta))\cdot\left(2\cdot\cos(\theta)^2 - 1\right)$ (C.2)

$dK1(\theta 1, \theta 2) := K1(\theta 1) - K1(\theta 2)$ (C.4)

$\phi := 0, .1 .. 2\cdot\pi$



c. The y value of dK(θ)



d. The x value of dK(θ)

Figure C.1: Plots (vs. posture and step size) of $\mathcal{K}(\theta)$ and $d\mathcal{K}(\theta)$

As the plots of figure C.1 demonstrate, the functions $dK_i$ are in fact linear, (in $\Delta\theta$) because the plots of figures C.1c and C.1d demonstrate that

$$dK_i(\varphi,\lambda\cdot d\theta) \approx \lambda\cdot dK_i(\varphi,d\theta)\,.$$

Furthermore, if we note that $dK_2$ is a maximum at $\phi = \pi$, then at that value we can pick the value

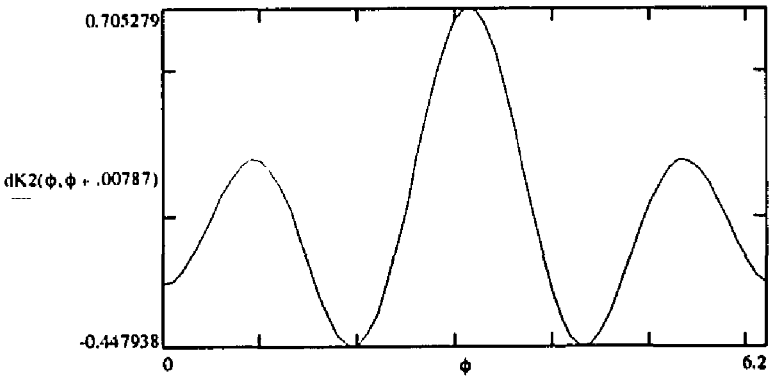
$$\left| \, dK_2(0.1) \right|_{\varphi=\pi} \approx 8.97745$$

off the plot of C.1c. Now by trial and error, at that value of $\phi=\pi$ , we find a value of y such that the differential step is a unit step, i.e. $dy \approx \sqrt{1/2}$ . Reading that value off the plot of figure C.2a, we seek a plot that satisfies:

$$\left| \, dK_2(y) \right|_{\varphi=\pi} \approx 0.707$$

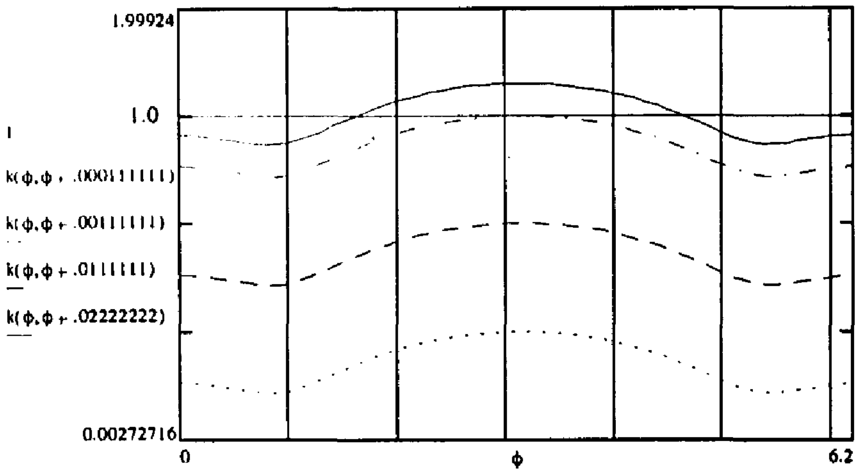
Such a plot exists for $\Delta\theta \approx 0.00787$ radians = 0.5°. The $dx$ plots give similar magnitudes.

Figure C.2b and C.3 plot the function $|dk(\theta)|$. *More correctly, it is the function* $|dk(\theta,\Delta\theta)|$. Figure C.2b is a log ordinate plot which in which it is clear that this function, $k(\theta1,\theta2)$ in equation C.5, is in fact nearly linear in $\Delta\theta$. Figure C.3 shows this function only in the domain from 45° to 180°, which is the region in which the articulator simulator usually operates. The objective here is to find that range of $\Delta\theta$ which over the domain keeps the step length as near 1.0 as possible in the hand space.



a. Containing the dθ to a Unit Length Hand Move

$$k(\theta 1,\theta 2) := \sqrt{\left(dK2(\theta 1,\theta 2)^2 + dK1(\theta 1,\theta 2)^2\right)} \qquad \text{(C.5)}$$



b. Log plot of dK for dθ giving a move less than or equal to unit length

Figure C.2: Plots of d$\mathcal{K}$for, at Most, Unit Length Hand Moves

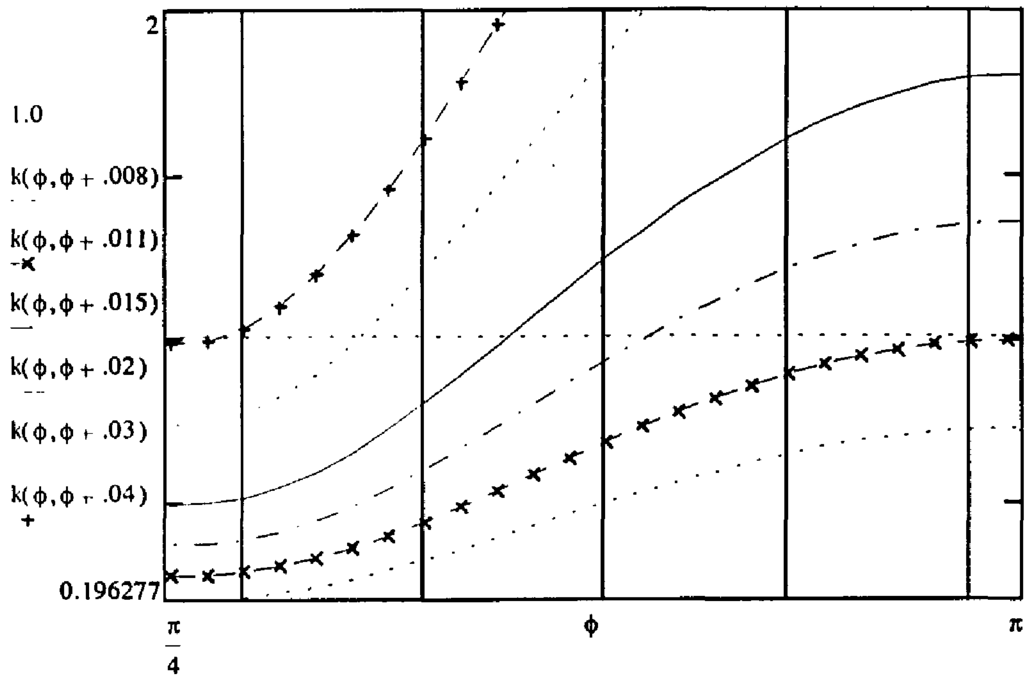
The limiting plots are the + hatched and x hatched plots in figure C.3, which correspond to, respectively, $\Delta\theta = 0.011$ and $\Delta\theta = 0.04$ radians, or $\Delta\theta = 0.64°$ to $2.29°$. These values correspond to

$$0.64° \sim 1 \textit{ codon unit and}$$

$$2.29° \sim 3 \textit{ codon units,}$$

so for 3 angles, an average of 2 cu/joint for an average of 6 cu's is probably a reasonable if slightly high estimate. But if $\Delta\theta \in (0.64°,...,2.29°)$, is the change in hand space direction cosines significant? Since the hand will be directed towards the target, it is reasonable to assume that the angle $\theta$ , of figure 4.111 (on page 111) is small, because the vector **s** should superimpose on **c** 151if goal directedness is achieved. This means the opposite angle, $\zeta$ , is even smaller yet, at least until the hand gets very near the target. At any rate, on average, it is probably safe to estimate that the most activity that will be seen in this measure is a single codon unit, perhaps, due to numerical roundoff and dithering in the attempt to keep the hand on a rectilinear path to the target. As the CMAC gains experience, the change in the direction cosines per path segment should approach zero. For the record, I *did* observe this in practice by recording and displaying changes in the direction cosine components during simulation and they showed only least significant bit changes between trajectory steps. Incidentally, this means that this pair of coordinates is a strong index for storing and separating classes of trajectories, because it effectively segregates trajectories in the state space by orientation of the hand space trajectory!

If the trajectory has significant curvature in the hand space, then the direction cosines will change significantly and affect generalization slew rate, which is exactly what is observed in the Appendix E plots: Whenever the system



For postures in the first quadrant, try to iteratively isolate values of dθ near a unit length hand move. This will give a bound for dθ that can be used in the arguments relative to a discussion of the effect of generalization and quantization in the input space.

Fig. C.3 | dK | for Unit Length Hand Moves in Quadrants I & II

is in early training convergence, the generalization slew rate is transiently very large, and until the hand space trajectory becomes linear, large variance of the slew rate persists. It is also interesting to notice that the slew rate for the six and nine jointed arms of figures E.3 through E.6 show larger variance in this metric than the 3 jointed arms. This latter can only be because the hand space trajectories have higher curvature for the longer arms. It can not be because of variance in the joint steps, because the stepsize control (AGC) module prevents this. Again, observation supports this because the longer arms *did* exhibit more hand space trajectory curvature than the three jointed arms.

So in toto, for the 3 jointed arm, we should see $\leq 6 + 1 = 7$ codon units change between steps, or $7/64 \times 100 \leq 11\%$, which is about what the plot of figure 4.10, ( on page 109), showed. Observing the other plots on that page and the previous page, note that by halving generalization or hand quantization, the effect is to double the generalization slew rate, again, exactly what the above analysis predicts.

## *APPENDIX D*

## *Patch Plots of Receptive Field Ranges*

In the pages that follow, I have included several plots of the range of the receptive fields of the ARTISTS CMAC under certain conditions, for varying generalization, memory size, and varying posture. The object is to make some conclusions to lend credibility to the reasons for success of the ARTISTS trajectory data storage method.

### *D.1* *Visualizing the CMAC Data*

It is obviously impossible to visualize the nonlinear mapping of even the simplest redundant manipulator handled by ARTISTS. The reason is that the input is at least 5 dimensional and the output at least 3 dimensional, (for the shortest redundant linkage). Figures D1 through D8 show examples of 3D surface patches, each of which is a representation of the output of the CMAC for a 3 jointed articulator trajectory planner. In each case, the assumption is made that the base angle, $\alpha$, and the first constrained joint, $\beta$, are free variables. The third joint, $\gamma$, is constrained to be equal to $\beta$, so it seems reasonable to view plots of the outputs, $\delta\alpha$, $\delta\beta$ and $\delta\gamma$ as functions of the 2 free variables. To generate the plots, $\alpha$ is incremented over a range of C codon units starting with $\alpha_0$-C/2 and proceeding up to $\alpha_0$+C/2, ($\alpha_0$ is the base angle setting corresponding to the figure in each case). For each setting of $\alpha$, both $\beta$ and $\gamma$ are incremented in parallel from respectively $\beta_0$ and $\gamma_0$, through the same sized range, and the output vector is plotted at each resultant point. The hand space direction cosines, $\delta x$ and $\delta y$ are held constant throughout each patch plot. The result shows some indication of the shape of the inverse kinemaitc function around the operating point defined

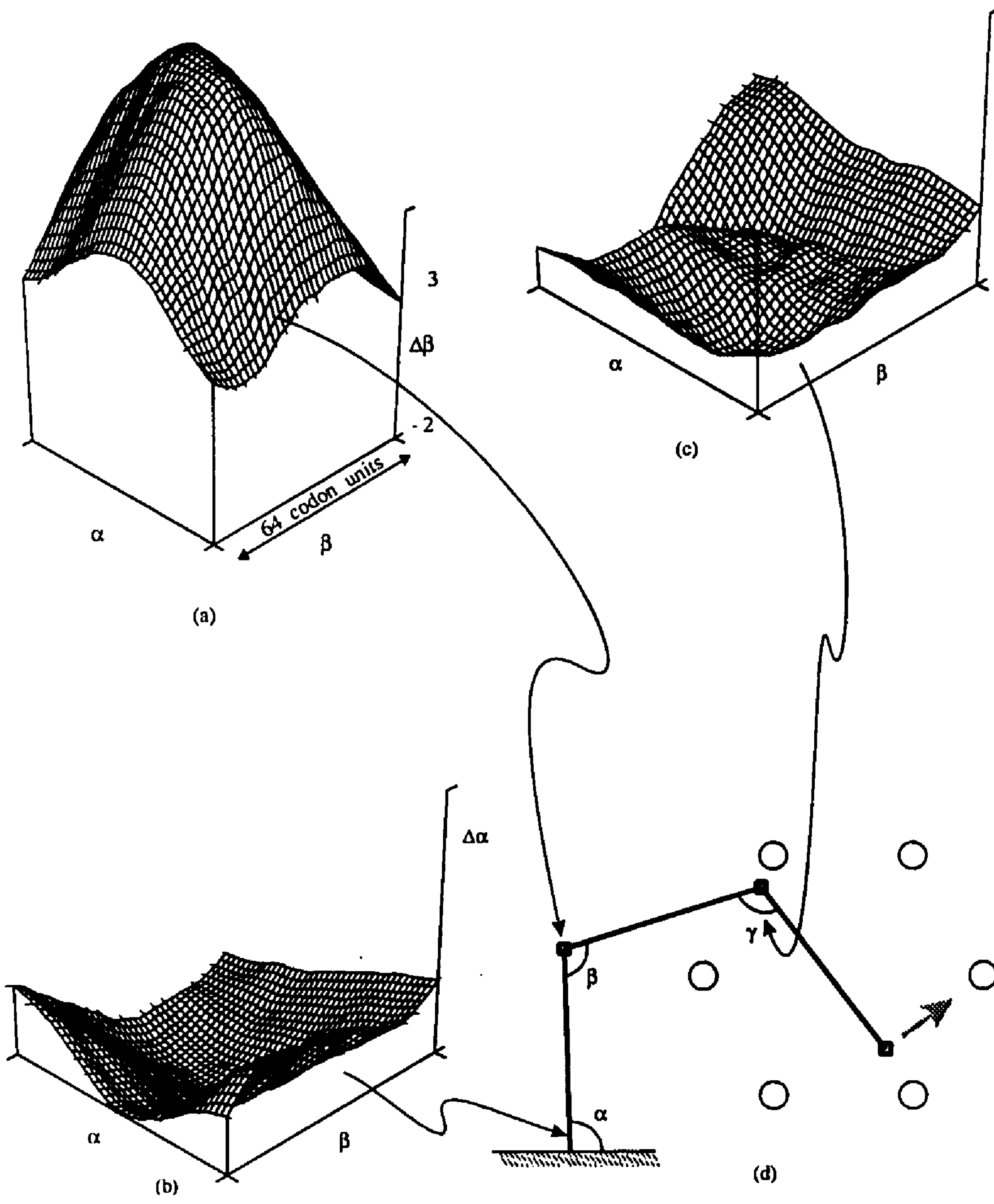


Figure D.1: Δ Angle Response Plot
as Function of α and β in an Intermediate
Posture (Posture #1). Generalization = 64

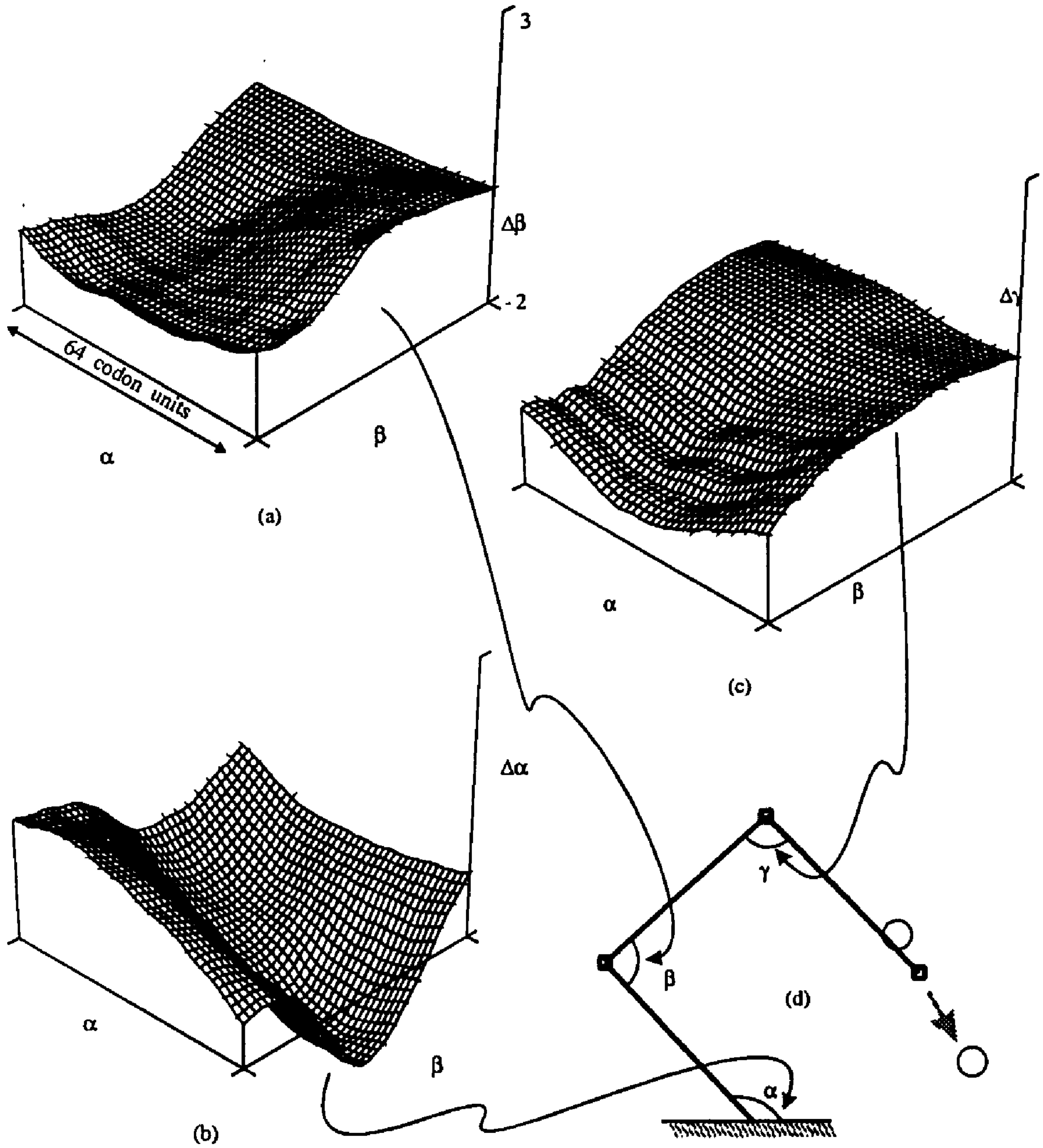


Figure D.2: Δ Angle Response Plot
as Function of α and β in a retracted
Posture. (Posture #2)

Generalization was 64, and a memory size of 20K vectors was entirely adequate.

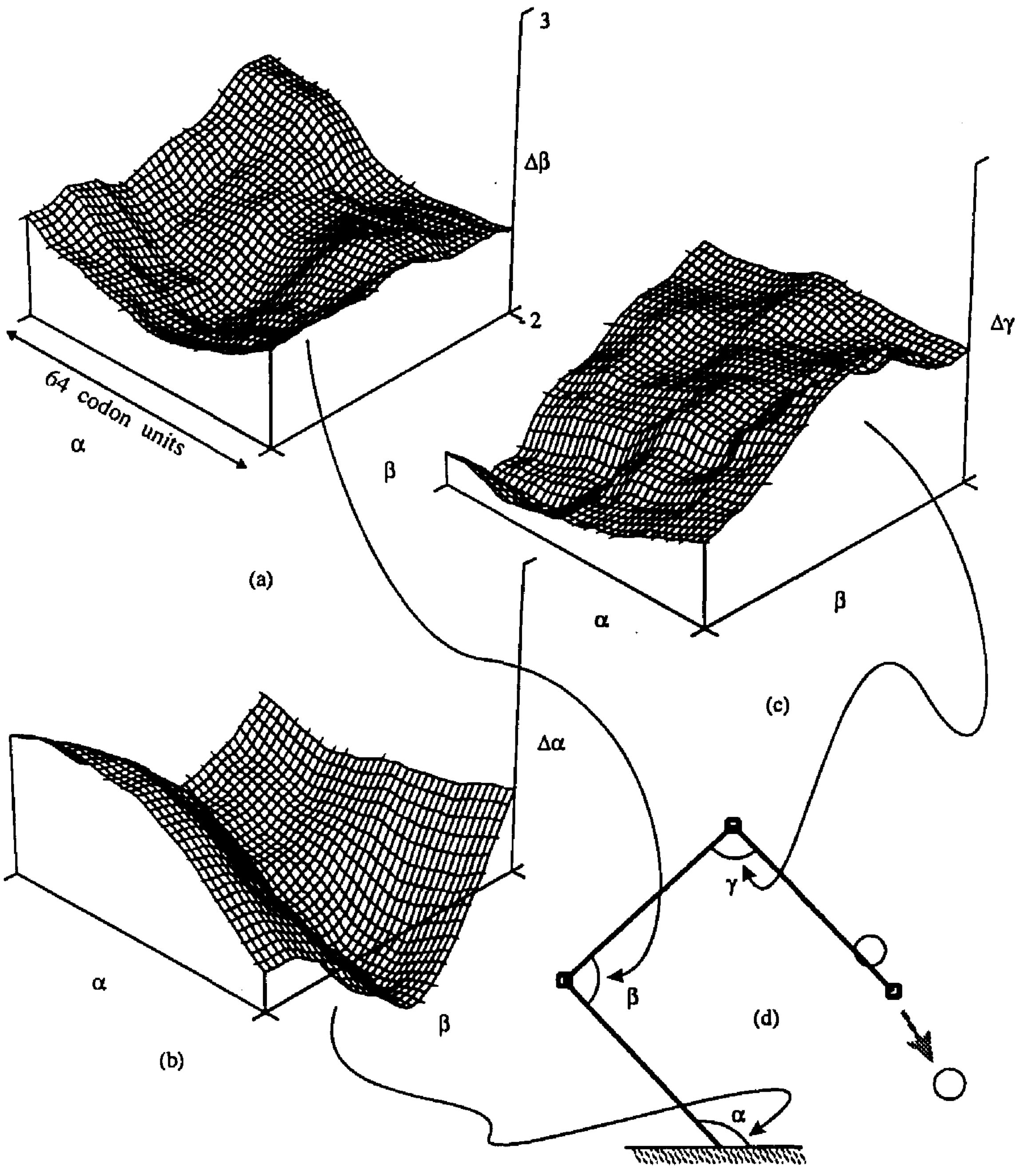


Figure D.3: Δ Angle Response Plot
as Function of α and β in a retracted
Posture. (Posture #2)

Generalization again was 64, but the small memory size of 4K caused hashing damage to be visible in a slightly more erratic shape of the surfaces. The overall effect was that performance, though degraded was gracefully so.

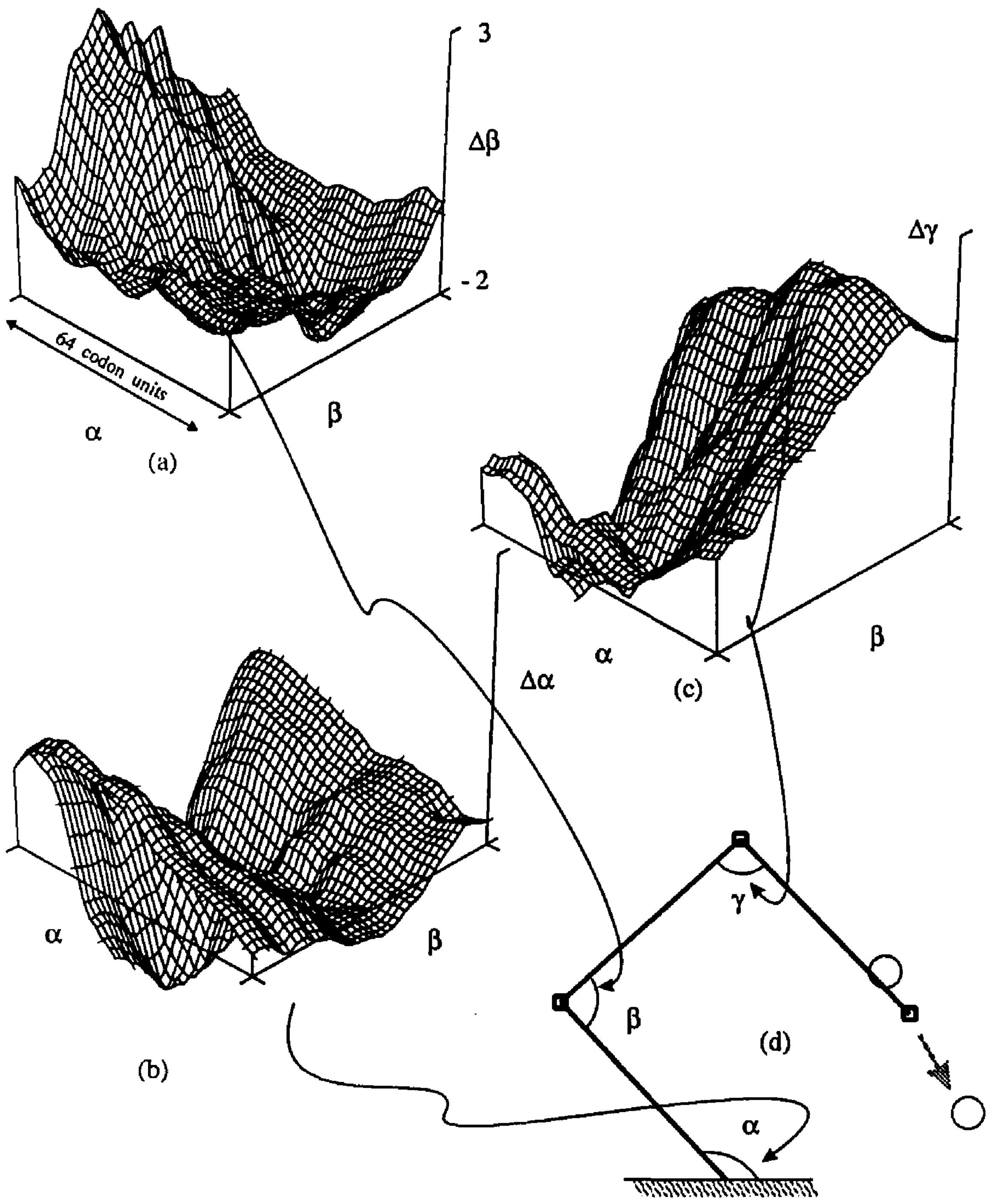


Figure D.4: Δ Angle Response Plot as Function of α and β in a retracted Posture. (Posture #2)

Here generalization was 64, but the memory size was only 3K vectors. Hashing collisions became a problem. It took a lot of training to get this result, but remarkably, performance was eventually successful.

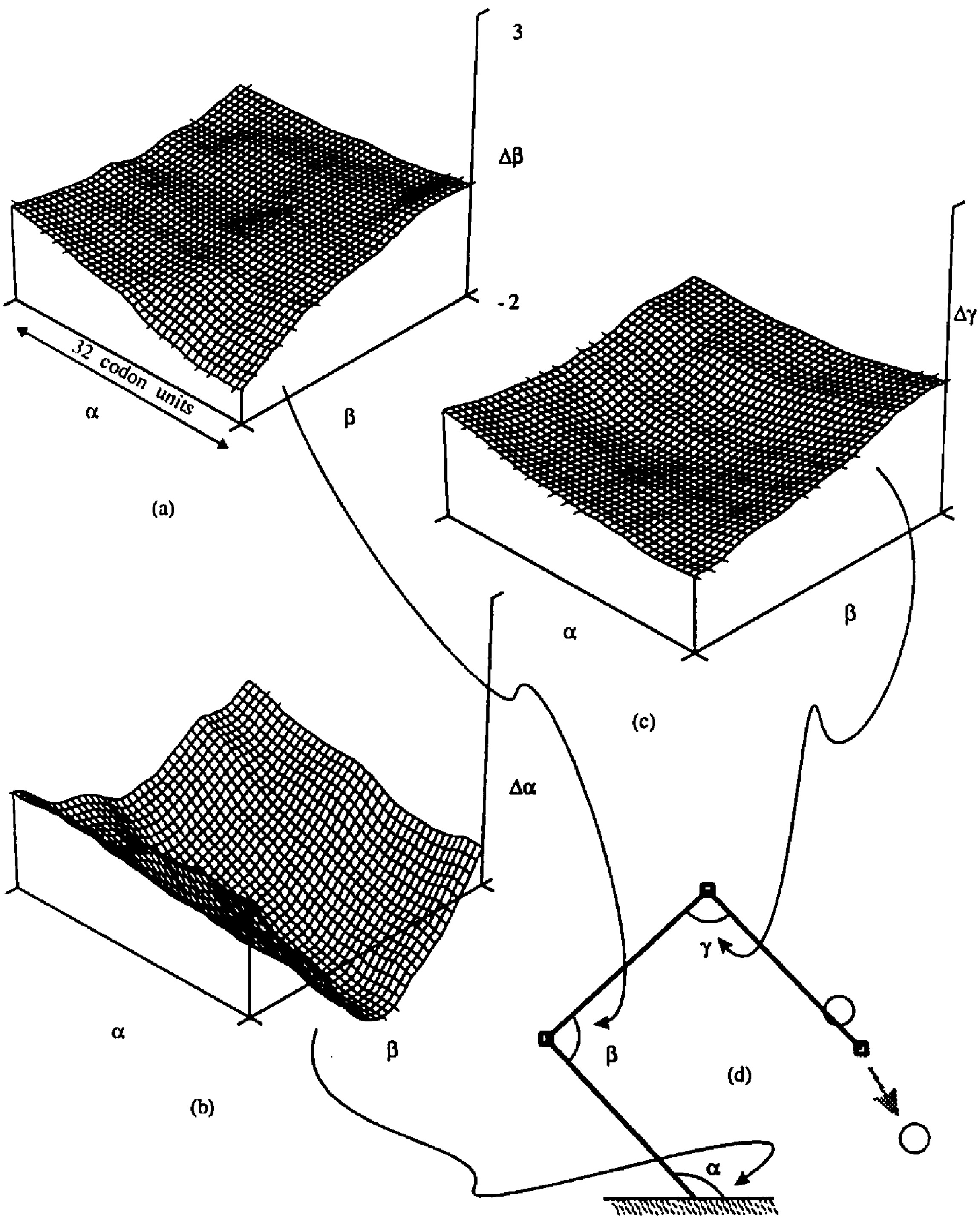


Figure D.5: Δ Angle Response Plot
as Function of α and β in a retracted
Posture. (Posture #2)

"Undergeneralization" for a CMAC with generalization = 32. Note that not as much of the nonlinear nature of the kinematics is captured in receptive field range. Learning is slow, approaching table lookup, and it is not as good at low pass filtering out hashing collision damage.

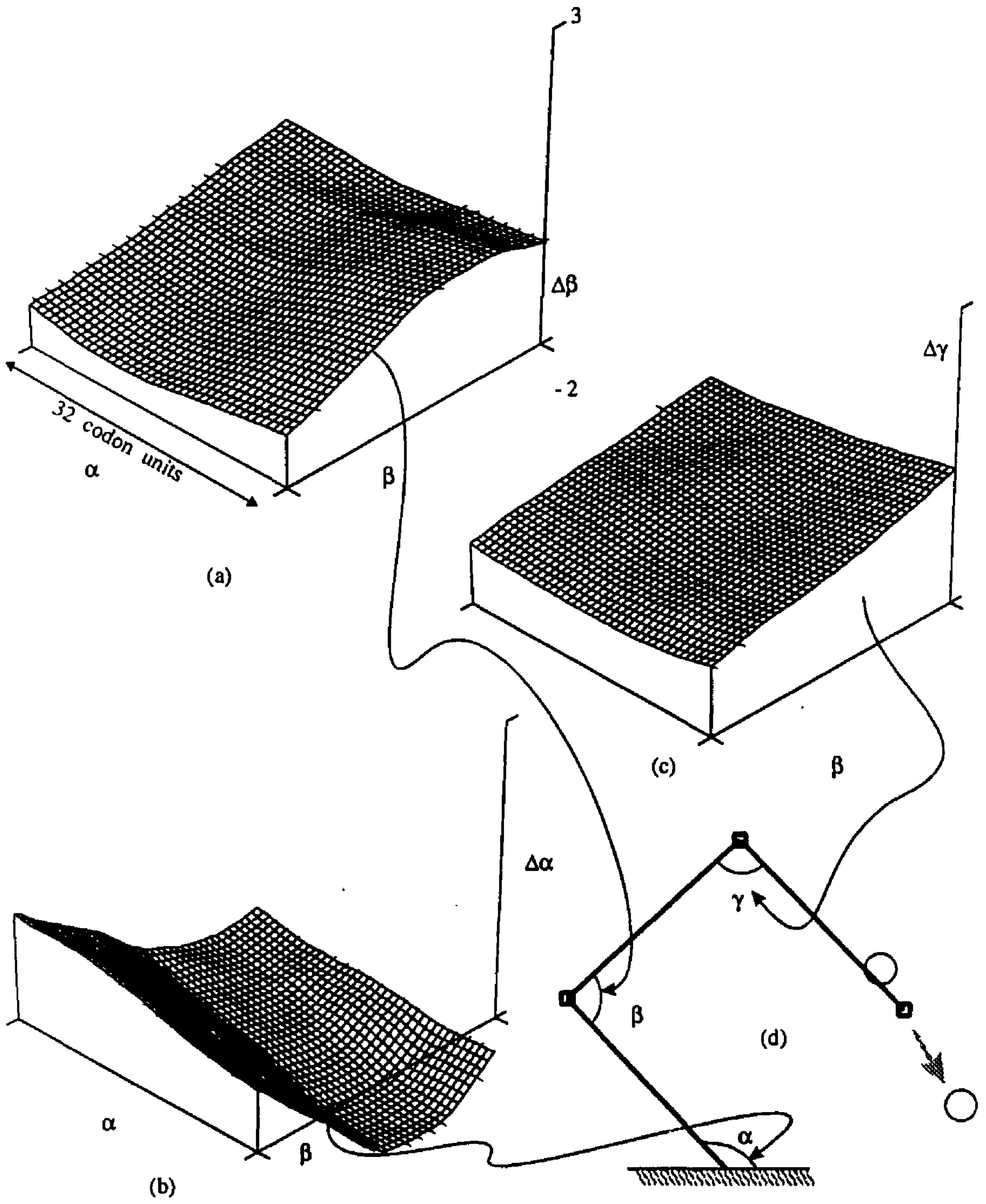


Figure D.6: Δ Angle Response Plot
as Function of α and β in a retracted
Posture. (Posture #2)

As in figure D.5, not as much nonlinearity shows in this patch plot. It, however is the result of plotting a half generalization width patch for a CMAC with a generalization = 64, so it subtends the same width as figure D.5. The relevant observation is that the plot is smoother. Any apparent shifting of the patches among figures D.2 through D.6 and most shape change is due to slight differences in the operating state space point.

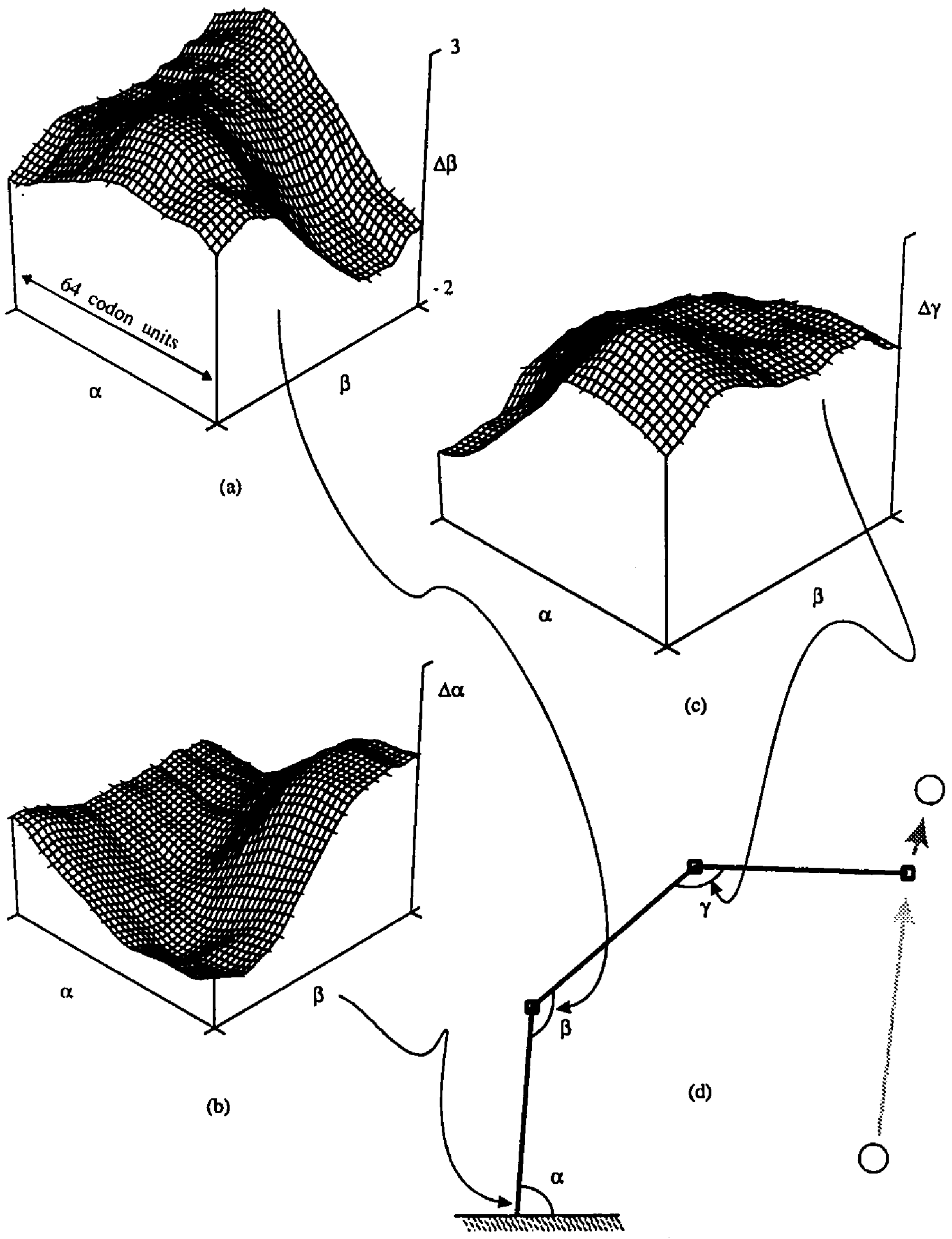


Figure D.7: Δ Angle Response Plot as Function of α and β in an extended Posture. (Posture #4)

Generalization of 64 and memory size of 20K vectors.

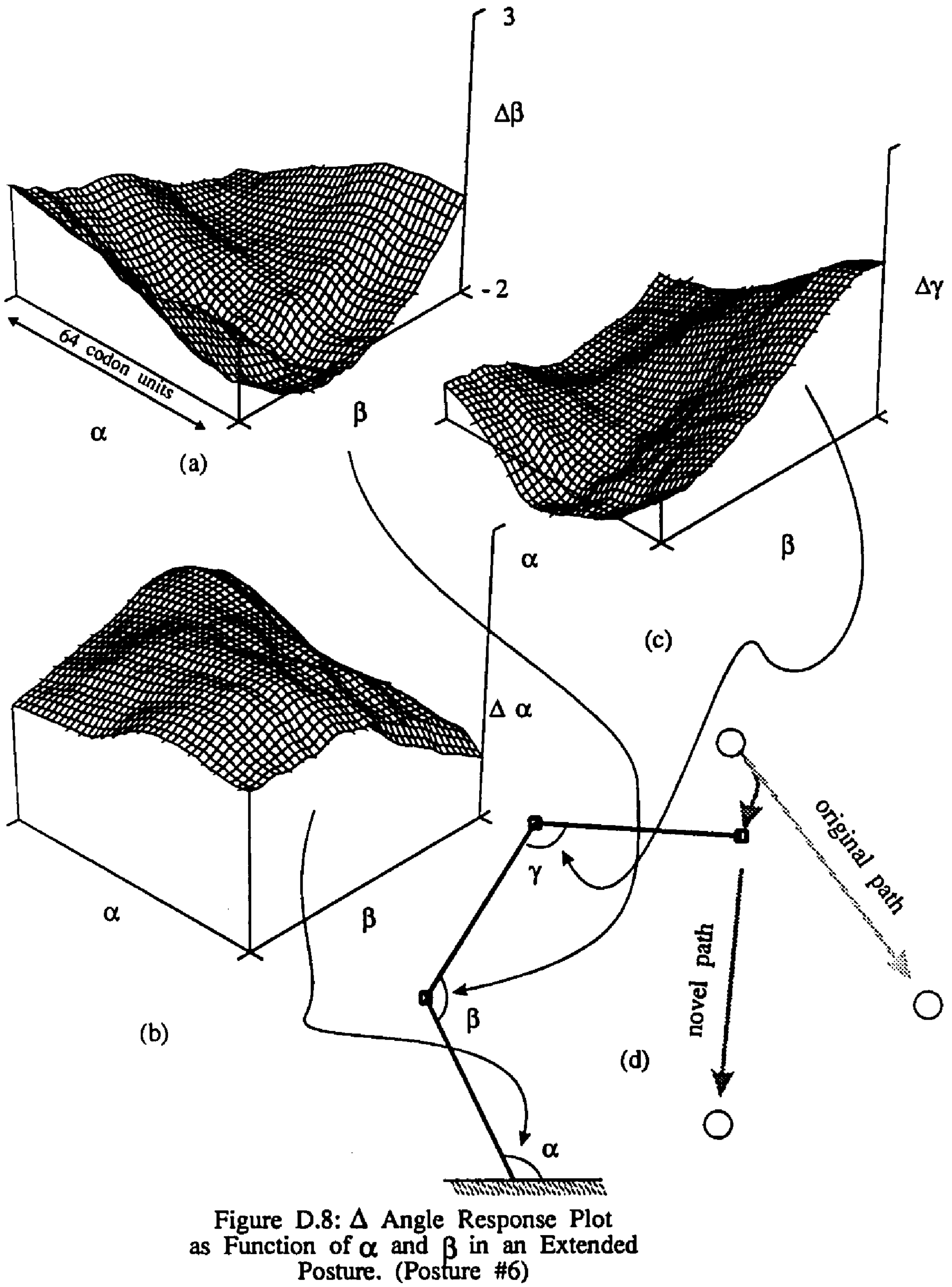


Figure D.8: Δ Angle Response Plot
as Function of $\alpha$ and $\beta$ in an Extended
Posture. (Posture #6)

This posture resulted from a diversion of the arm from a well trained path segment to a completely novel trajectory. Generalization was 64 and memory size 20K vectors. "Texture" in this "novel" posture is much the same as in well trained postures.

by the posture shown in the figure. By varying $\alpha$ and $\beta$ and observing the magnitude of the vertical axis, the angular displacements on each joint required to maintain a hand move of ($\delta x$, $\delta y$) can be read off the vertical axes of the three patch plots in each picture.

The patch plots cover an area of state space approximately equal to an actual receptive field's coverage. It is important to note that the functions generated, (which must be a close approximation of the inverse kinematics, since the system functions appropriately), have at most one "hill or valley" in the patch. This means that spatial frequencies > 1/C are not strongly represented in the functions. We know from Carter ['90] and Segee ['92] that if this were not the case, CMAC learning would be slow. Given the linear tapered window, the one humped or one valleyed topology of the surface fits well the receptive field contour. I would have been worried to find many peaks and valleys in a receptive field extent, because I would expect that they would cause the CMAC mapping to be potentially unstable, or at best slow to converge, due to CMAC averaging, which would force those many peaks and valleys to change rapidly during training in an effort to force the average at each iteration to look like the observed response.

It must be noted that these patch plots are *not* receptive field descriptions. A C = 64 patch represents a sampling of 1600 sample points each, because the patch plot software is limited to 40x40 plots. These samples are evenly distributed over 4096 virtual state space points, (a 64x64 codon square). Each point is the summation of 64 weights that are regularly, sparsely and uniformly distibuted over hypercube regions centered around each of the 1600 input state space points.

So the functions represented in each surface plot are the conglomerate effect of many receptive fields. It should just be noted that the influence of just the receptive field associated with the center of each plot extends over the plot, with declining influence toward the periphery. The observation I expected to see in these plots was that consistently shaped surfaces with a topological simplicity would emerge after a minimal amount of training, and that they would be repeatable. This expectation was born out conclusively by the following observations about the plots:

## *D.2* ***Discussion of Results of VisualizationExperiments***

It is obvious that figures D.2 through D.6 depict the same function, albeit with some varying degrees of distortion or scale; this is appropriate since the state space input for each was similar, but not precisely the same. The reason is that for D.5 and D.6, separate experiments with differing generalization were used. For D.3 and D.4 differing physical memory sizes were used. In D.2 and D.6, the same experimental setup was used, but I manually interrupted the program along subsequent trajectories to snapshot patches of different extents. Figure D.6 is an attempt to display a receptive field sized region of influence that is the size of a C=32 CMAC's receptive field, but is generated by a C=64 CMAC. The observation here is that the C=64 CMAC gives a much smoother representation over the same state region.

So it is possible to visualize in a limited fashion the mapping in a high dimensional CMAC. It is further possible to conclude that the reason C=64 works better than C=32 is that for C=32, the function representation is less smooth, is less able to low pass filter out the noise of hashing collisions, and approaches more closely table lookup, with a much more localized receptive field extent and thus a more nearly linear function under each receptive field as can be confirmed by viewing figure D.5.

These plots are characteristic. I plotted many more and noted that they all told similar stories. Though they had widely differing shapes, all the shapes were of single bumps or valleys or at worse, mild saddle points like figure D.8.a and b. Figure D.2 is overall smoother than D.5 and D.5 took well over twice the training time to achieve its result than D.2. All of this is reasonable for comparing C=64 and C=32.

D.8 is a very interesting example, demonstrating generalization in the "traditional" sense. This surface shows little qualitative difference from D.2, i.e., it is smooth and reasonable in shape, though it is a patch plot taken the first time the system executed the trajectory. The fact that the trajectory executed without any divergence from the expected path indicates that generalization was strong, and the function in D.8 looks very much as it will after ultimate convergence.

Finally the sequence D.2, D.3, D.4 was extremely interesting. D.2 performed best of all the experiments. When its memory size was reduced to 4,000 vectors memory saturation was over 30%, and the toll hashing took is evident in figure D.3. Note that its shape is still recognizable, but quite irregular and distorted. With a further reduction in memory size, (figure D.4), performance became quite unacceptable, and the surface quite noisy, with memory saturation around 50%. Surprisingly, this last distorted model is still capable of learning. If we leave it for 1000 iterations, we get the results shown near the left side of figure 4.10 on page 109. Its error statistics in the steady state are about the same as the setup with C=64 an 8000 memory vectors. All models that these patch plots represent were trained very superficially. All were trained on less than 200 path segments, and some on only 30 or so.

## D.3 *Conclusions From Visualization Experiments*

The conclusions I derive from these observations are that

- That ARTISTS is robust with respect to noise (as a function of memory size) as argued in section 4.8.7 is further supported by these plots
- The parametrization with C=64 does in fact seem reasonable, as was discussed in section 4.8.6.
- The simple shape of the control surfaces subtended by receptive fields is conducive to a consideration of CMAC as a reasonable spatially distributed generalization of traditional non-linear control by linearization of equations at operating points. See section 6.2.

# *APPENDIX E*

## *Raw Error Metrics Data From ARTISTS Trials*

In figures E.1 tthrough E.6 are examples of the raw error metric data derived from various trials of the ARTISTS system, with constraint satisfaction but no obstacle avoidance. The body of the thesis refers to these figures with appropriate commentary. The general features to notice about these figures is that they represent successful parameterizations.

The error metrics for figures E.1 and E.2 was smaller than for any of the parameterization trials. The postural error metric falls off to near zero within the first 100 path segments. Heuristic step density drops off to exactly 0 in the same time frame.

Figures E.3 through E.5 shows the error metrics plotted for 6 jointed arm experiment with non-repetitive targets. The postural error metric falls off to a small value, but not as small as with the 3 jointed, arm, which is expected. This was a casually chosen parameterization. It is possible, indeed likely that a better one exists. Note in figure E.4, around epoch 1600 the error metrics increase significantly, and heuristic help is required for about 100 epochs. What happened there was that while the program ran unattended, some randomly wandering targets moved off into regions where the arm could not reach, because they were more than the sum of the link lengths away from the base. The system was not designed to handle such a problem, and performance was not surprisingly very poor  for a substantial period. It was gratifying to note, though, that when I intervened and collected all the targets, moving them back to within the arm's field of reach, the system recovered nicely. The only adverse effect was that an additional few hundred vectors of memory were consumed. After that point the error metrics went back to very low levels and memory vectors used remained constant for the rest of the 3000 epochs.

Figure E.6 shows the result of a non-repetitive 9 jointed arm trial, which shows postural error of roughly the same magnitude as the 6 jointed arm. Note that scale for postural error is changed for each of the three cases of 3, 6 and 9 joints, to account for the extra number of joints. For instance, the scale of 0 to 40 (degrees*joints/pathlength) for the 9 jointed arm is 4 times larger than for the 3 jointed arm, because there are 4 times as many constrained joints over which to compute the error. The other metrics are not so scaled.

The actual units and scaling of these values is of little importance. They were only recorded

- To show that they in general decreased over time, and
- To provide a *relative* measure for discerning the best parameterization for the 3 jointed arm.

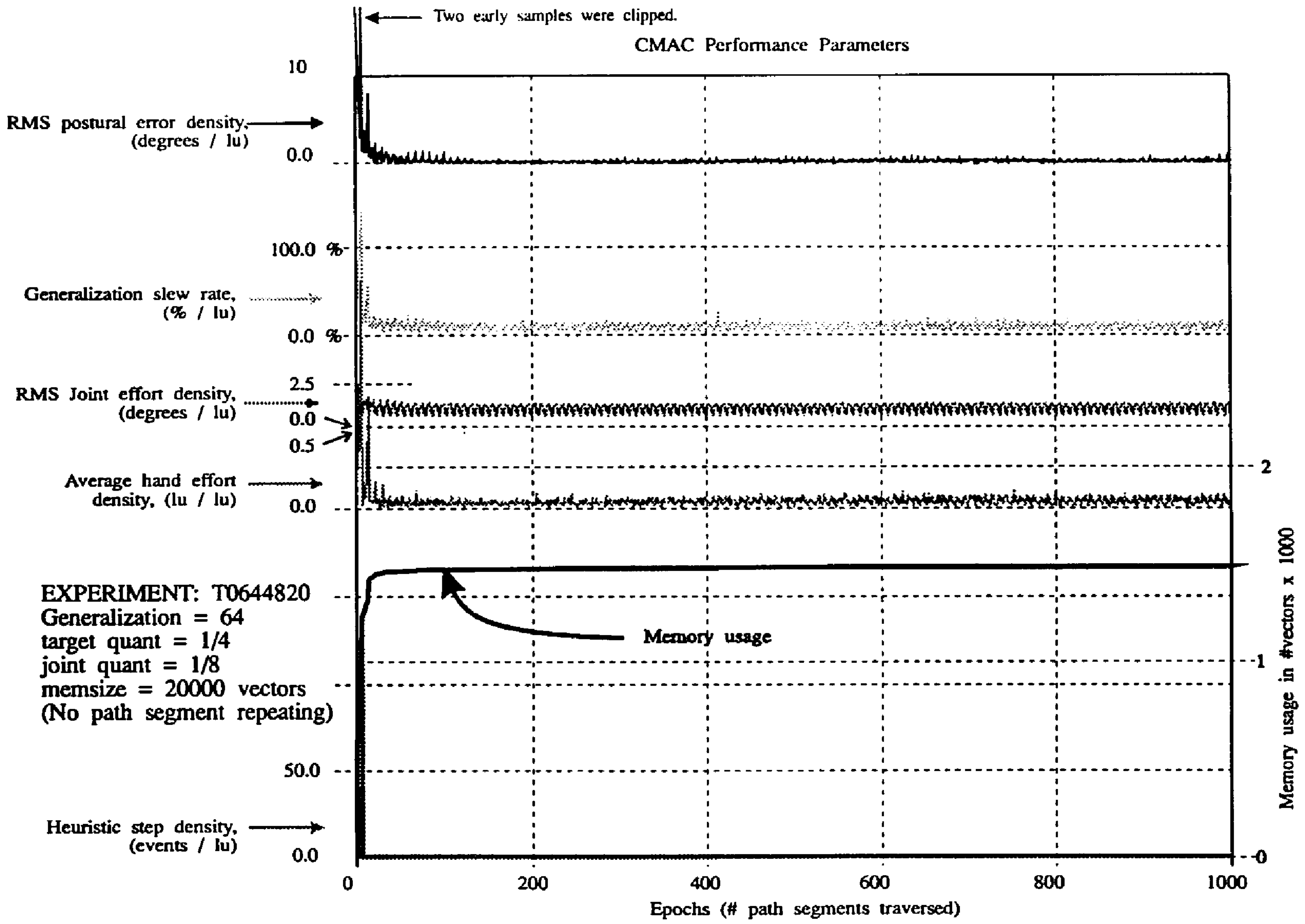


Figure E.1: Error data from Best Parameterization Trial

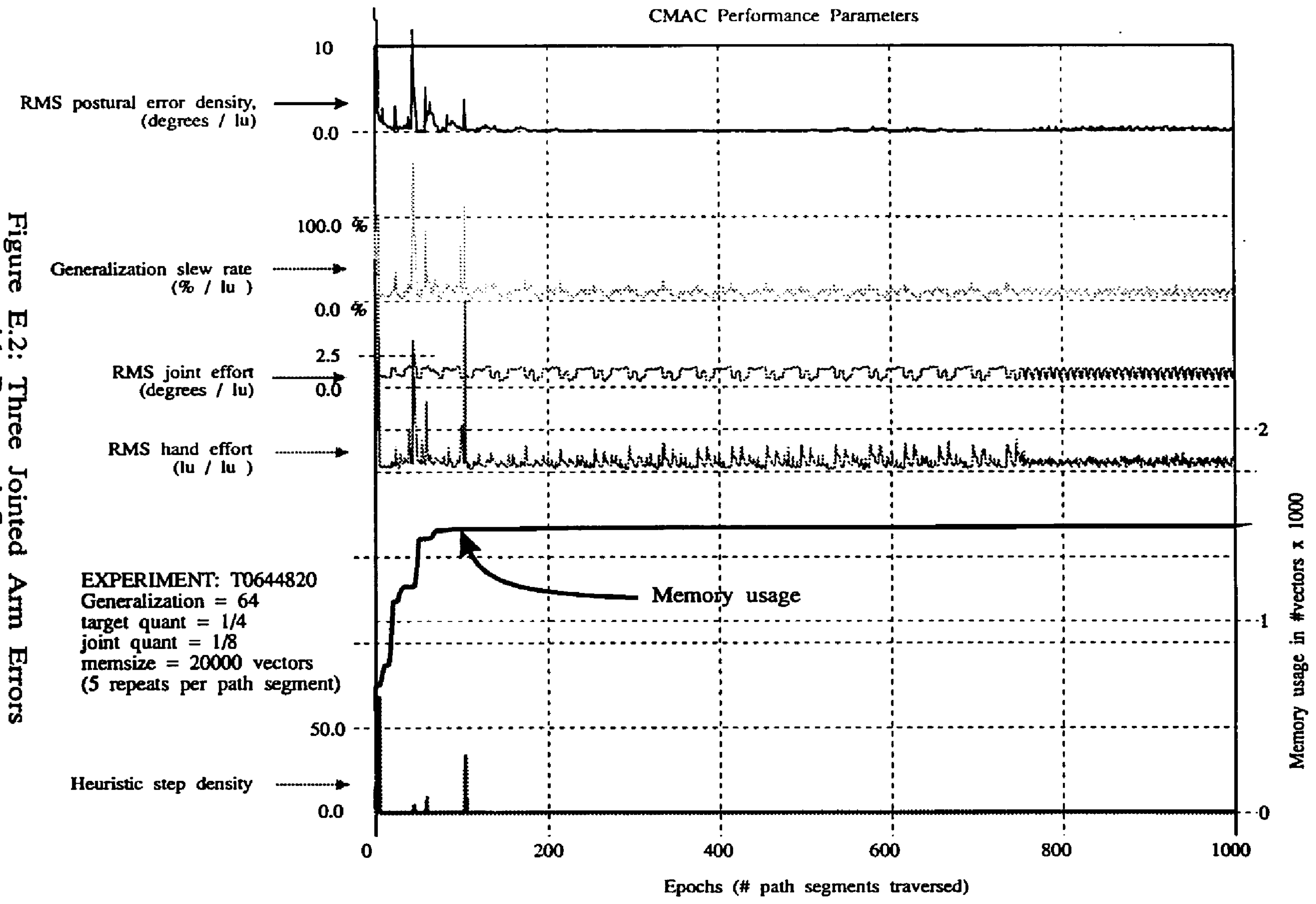


Figure E.2: Three Jointed Arm Errors with Repeated Segments

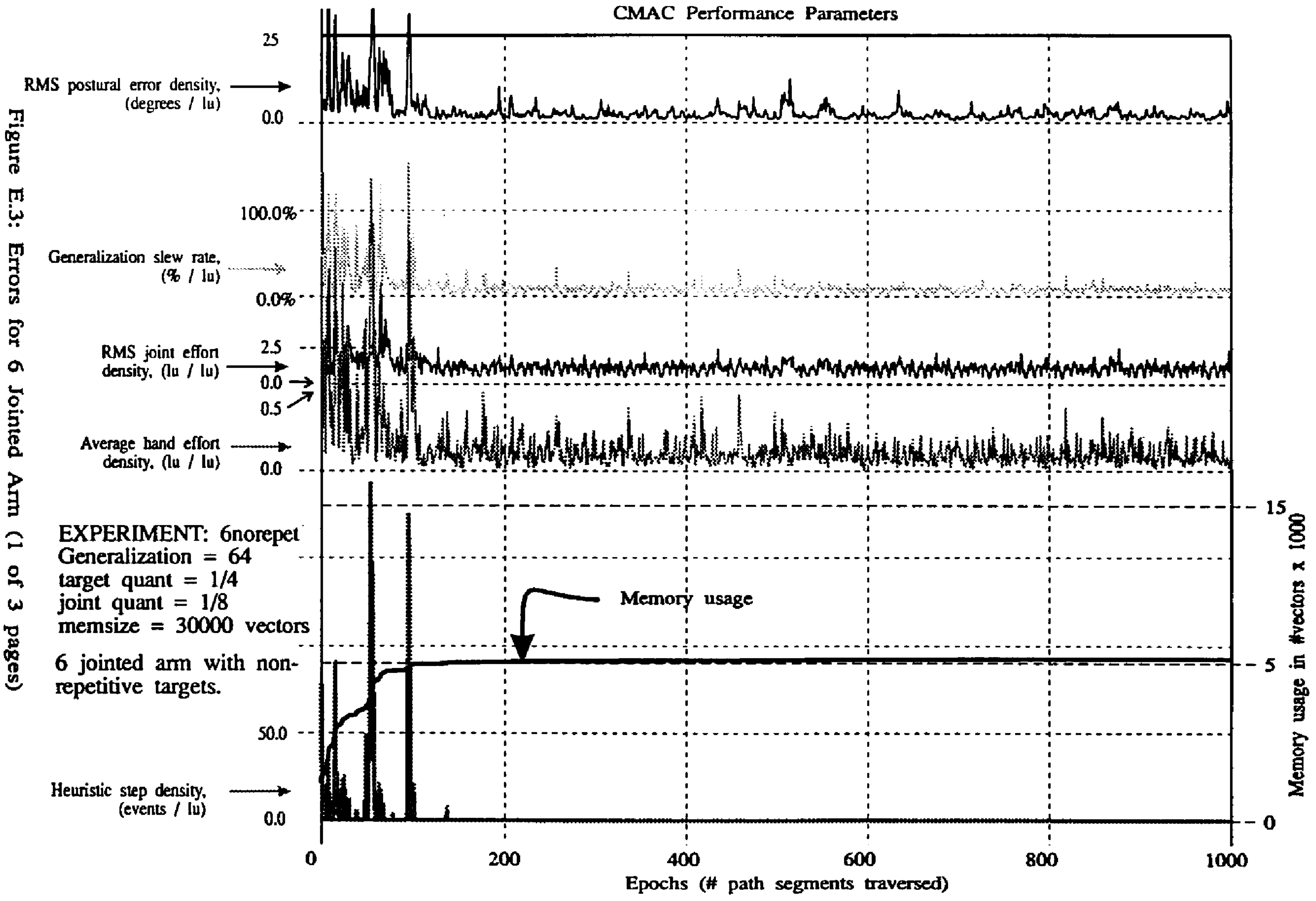


Figure E.3: Errors for 6 Jointed Arm (1 of 3 pages)

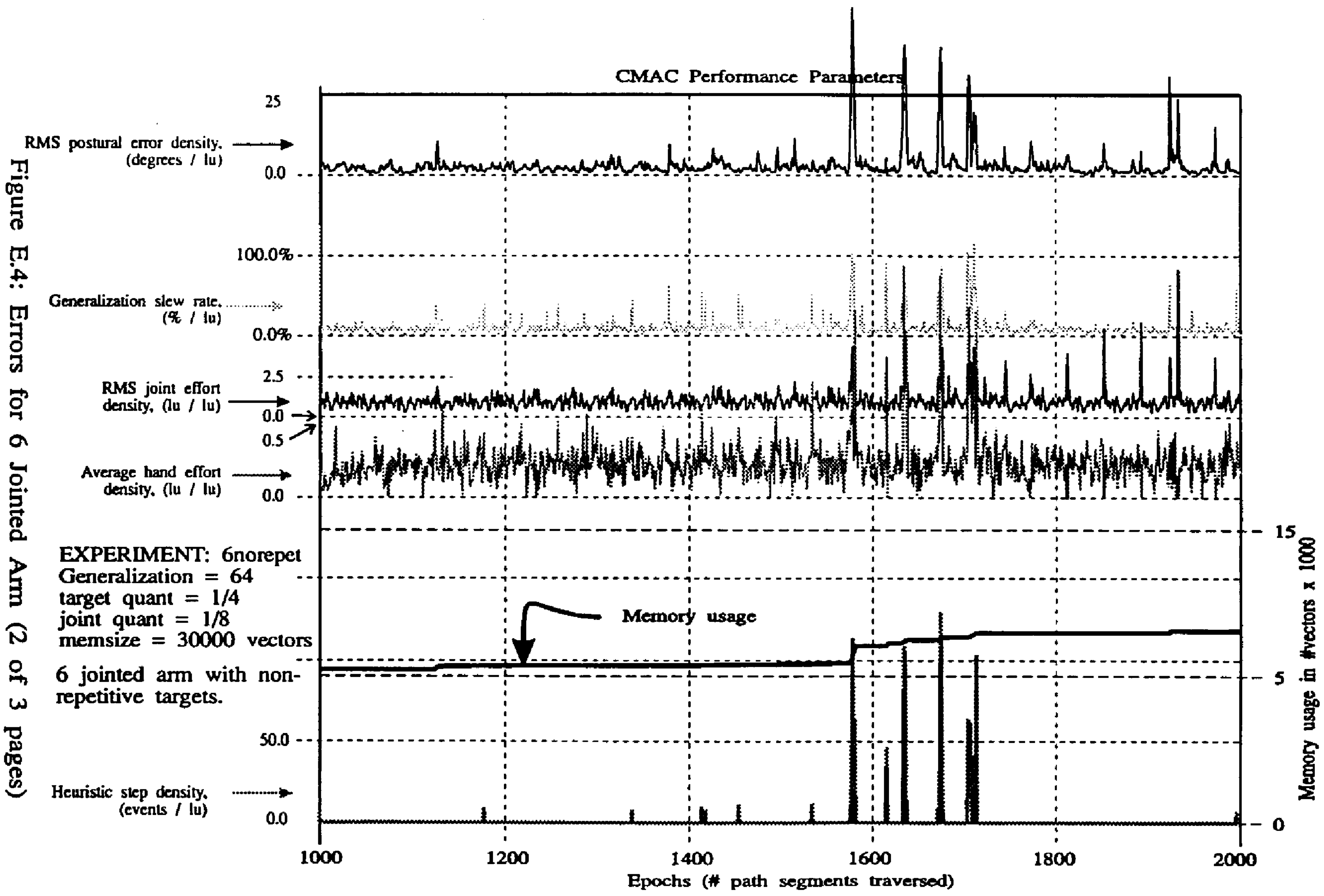


Figure E.4: Errors for 6 Jointed Arm (2 of 3 pages)

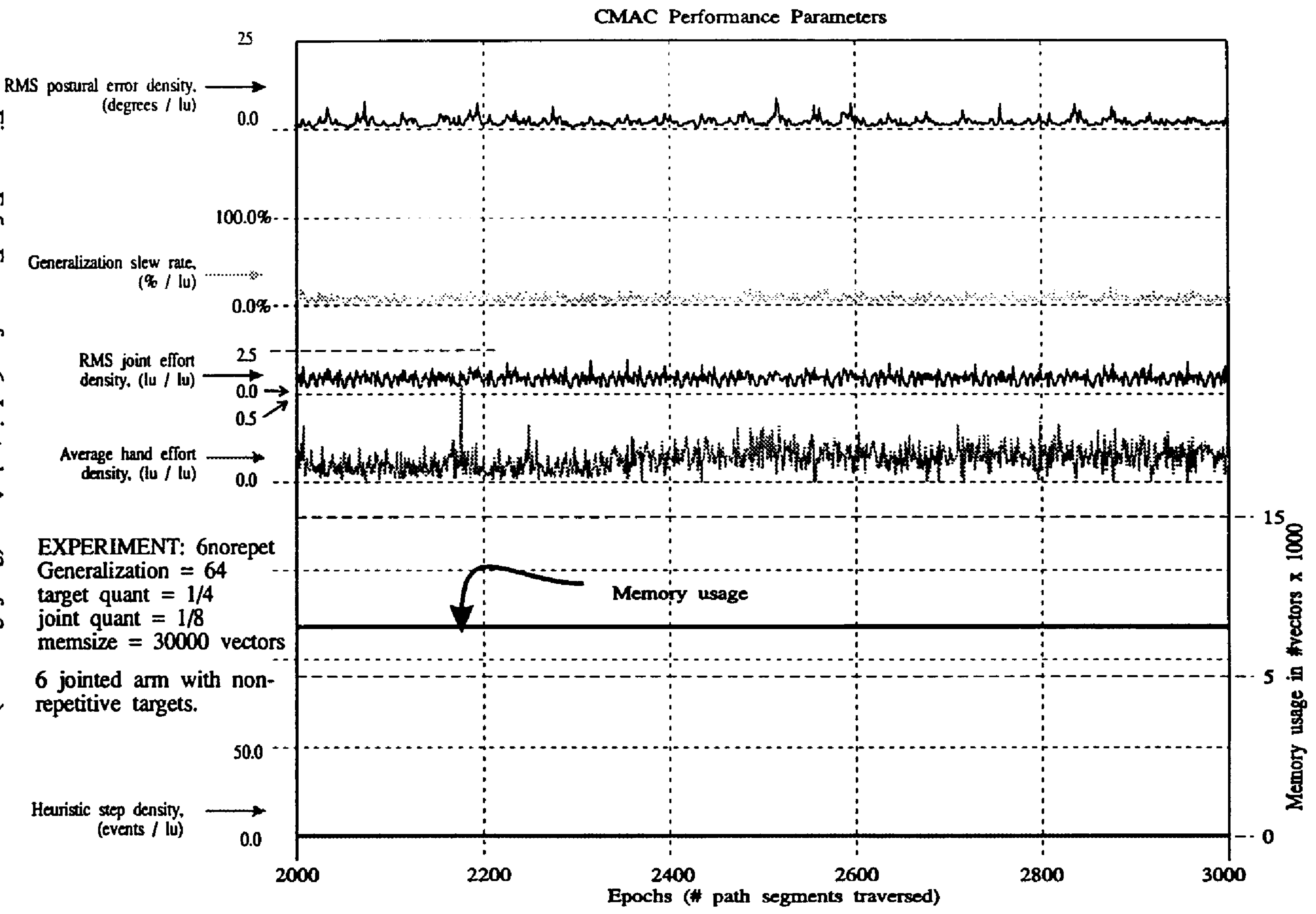
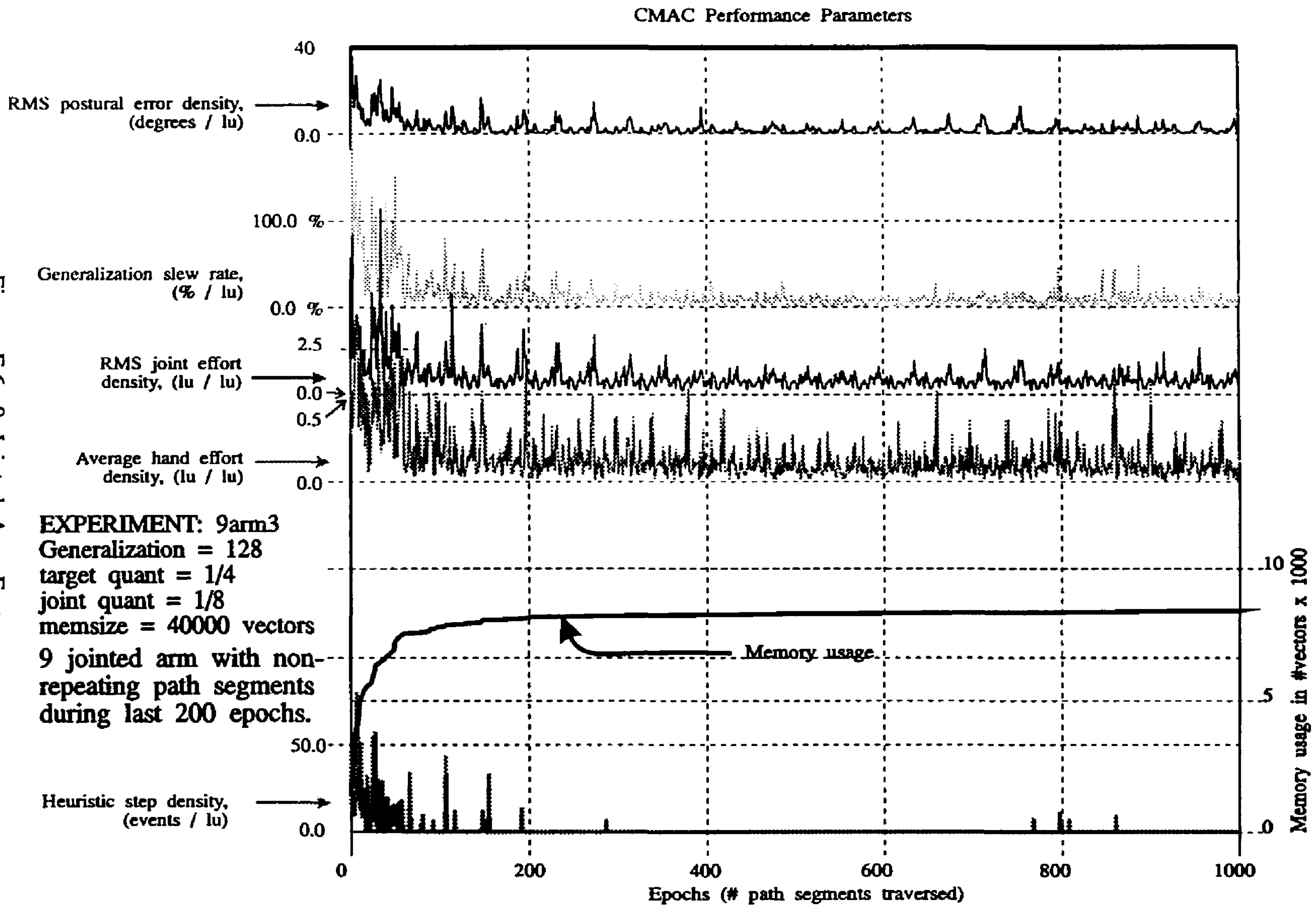


Figure E.5: Errors for 6 Jointed Arm (3 of 3 pages)



**EXPERIMENT: 9arm3**
**Generalization = 128**
**target quant = 1/4**
**joint quant = 1/8**
**memsize = 40000 vectors**
**9 jointed arm with non-repeating path segments during last 200 epochs.**

Figure E.6: 9 Jointed Arm Errors

# *APPENDIX F*

# *ARTFORMS User's Manual*

Version 6.3 11/5/92 Copyright 1988, 1989, 1990, 1991, 1992 by Frank Rudolph at the University of New Hampshire Robotics Lab. All rights reserved.

Before you do anything: type CMAC_SW to invoke the CMAC TSR program.

ARTFORMS will not do anything until that is resident, and in fact it WILL crash your system without it!

NOTE: This software will only work on an MS-DOS 80386 machine.

After CMAC_SW is running, execute ARTFORMS or ARM.BAT.

## *Part I: Keystrokes During Program Execution:*

(See the file checkey.c; It contains a dispatch table for these keystroke responses.)

- = suppress random heuristic steps.

+ = enable random heuristic steps.

? = HELP.

0 = Set random step mode for the heuristic step generator.

1 = heuristic mode 1: suggest equal magnitude angles but randomly select CW or CCW joint rotation.

2 = Mode 2: different magnitudes at each joint, but rotation direction same for all joints at each step.

3 = Mode 3: Like mode 1 WRT sign, but magnitudes of joint steps are ascending.

4 = Mode 4: Random magnitude and orientation of joint perturbations, but most change is concentrated at the base, with the rest of the energy distributed with increasing weight towards the hand.

5 = Mode 5: Like mode 4, but with random sign at the base.

6 = Berkinblitt synergy based step generator.

Biases for the heuristic random steps:

a = accumulate negative bias to alpha, (base).

A = accumulate positive bias to alpha.

b = accumulate negative bias to beta, (shoulder).

B = accumulate positive bias to beta.

g = accumulate negative bias to gamma, (elbow).

G = accumulate positive bias to gamma.

These invlove only the first 3 of up to NUM_JOINTS angles. The actual number of joints is specified by num_joints which must be < or equal to NUM_JOINTS. To specify variable values without recompiling, use the arm.ini file. There are over 50 variables that can be adjusted thusly. See parminit.c for the variable initialization parser. See Part II, below, the ARM.INI file description.

c, C = clear screen.

D = turn on debug output (i.e. the value of response vector before and

after learning).

d = turn off debug output

E, e = unused

s, S = "slow", i.e. draw arm each cycle

f, F = "fast", i.e. suspend drawing of arm

H, h = Return arm to "HOME" position and restart the target set.

i = re-read the ARM.INI initialization file. (Restore the system parameters to the defaults as they existed at the start of the program).

^i = save the ARM.INI initialization file. Save new startup parameter settings in a *.ini file.

K, k = Kill the CMAC memory; you will be prompted to tell the system which CMAC to erase, if there are more than one.

L, l = unused

M, m: Invoke the mouse driven target manager:

Once in the target manager:

LEFT BUTTON = add a target.

RIGHT BUTTON = delete a target.

MIDDLE BUTTON = drag a target to a new position.

ALL buttons = delete all the targets and insert new one at the mouse cursor.

LEFT+RIGHT places an "exterior" obstacle at the cursor.

ESC = leave the target manager.

N = unused

n = ?

O, o = Open script file output.

p = suppress joint space dot Product heuristic retraining.

P = enable dot Product heuristic retraining.

Q, q = quit the program (in an orderly fashion).

r, R = restart current path.

t, T = Type to the terminal the number of non zero cells in the physical memory. This is a rough measure of the level of saturation of the CMAC memory.

U = unused

u = ?

V = unused

^ = increase the value of the first constraint coefficient; set the 2d one to 1/first_one.

v = decrease the value of the first constraint coefficient; set the 2d one to 1/first_one.

W, w = Install ratchet conditions (set max/min angles to prevent curling into a kink and to prevent reversal of curvature).

X, x = unused

Y, y = unused

Z, z = Draw 3 patch plots in PostScript files p1.eps, p2.eps and p3.eps, which show the shapes of the direct inverse function over the span of one receptive field. The height or z value of the patches is in each case a function of angles $\alpha$ and $\beta$, which vary from the "current" value to the current value +/- C/2 codon units. The three z plots are then, respectively, the values of $\Delta\alpha$, $\Delta\beta$ and $\Delta\gamma$.

*FUNCTION KEYS:*

F1 = Constant speed target moves. (constant at the last speed achieved)

F3 = Suppress retraining heuristic. This is a global equivalent to training the CMAC Training Inhibitor to a constant 1 through all state space. See the data flow diagram in CHapter 3 for local obstacle avoidance.

F4 = Unsupresses the above F3 feature.

F6 = Spawn a DOS window

F7 = Record arm moves from the current position.

F8 = Playback the recorded arm moves.

F9 = Radial (sequential reaching) trajectory formation.

F10 = Daisy-chained trajectory formation.

The arrows move the target around up down, left and right as well as vertically. The arrows accelerate the longer they are used. To go back to slowest speed, simply hit any other key. (Mouse left right up down ok too)

You may also use the joystick to move the target. To do this, depress the pushbutton and move the joystick for target x/y displacement.

Movement of the arm by the joystick is done by stopping the arm via SPACEBAR and moving the joystick. Without the trigger pulled move the base and shoulder joints. With the trigger pulled, move the shoulder and elbow joints.

Depress both the joystick buttons simultaneously causes the joystick to be recalibrated.

Space bar = stop moving the arm. (all other functions continue)

Space bar also resumes the arm movement.

*Control and alt keys:*

^A = Display the links.

alt-A = Suppress display of the links (just joints will show).

^B = ?

^C = terminate program (abort)

^D = Draw the arm posture for the next control cycle to the postscript output file.

^E = Draw the arm posture for the next target position into the PostScript file.

^F = Flash arm segments on screen.

^G = unused

^H = unused

^I = See under 'i', above.

^J = Show the joints on the screen.

alt-J = Flash the joints on screen once.

^K = unused

^L = unused

^M = unused

^N = Normalize the direction to target vector.

alt-N = Don't normalize the direction to target vector.

^O = unused

alt-O = Reinforcement signal (OBSTACLE)

^P = DONT PUSH THIS ONE. IT ACTIVATES THE PRINTER (DOS FEATURE, REMEMBER?).

alt-P = Purge dump of the last 100 gradient descent errors (in hand space).

^Q = unused

^R = Random target moves (the current target jumps around randomly using Alt-R = no random moves the last length vector defined by the accelerating target algorithm.)

^S = scroll lock (DOS provided)

alt-S = toggle sound on/off.

^T = Fast Target (accelerating)

Alt-T = Slow Target (non accelerating)

^U = unused

^V = unused

^W = unused

^X = unused

^Y = unused

^Z = unused

^PgUp = save or "upload" the current control CMAC

^PgDn = restore or "download" an old control CMAC

^Home = Arm to HOME position and first target

*Mouse commands:*

| | |
|---|---|
| middle button | = Current arm position is start position |
| left button | = Turn Arm on |
| right button | = Turn Arm off |
| left+middle | = flash arm once |
| right+middle | = flash joints |
| all buttons | = toggle joints on/off |

The target can be moved with the mouse similarly to the method described under the arrow keys above.

## *Part II: The Initialization File.*

The following describes most of the program variables that can be specified in the ARM.INI file. This entire file is parsed in by the TSRARM program in the first stages of execution to define the articulator arm being simulated, and some parameters about the learning system.

CAUTION: Don't put comments in the actual ARM.INI file. Some of these parameters are NOT optional, and the TSRARM program will not execute if they aren't specified. If a required parameter is not specified in the ARM.INI file, ARTFORMS will abort and type an error message specifying which one was ommitted. Some parameters must be specified before others; e.g. the value num_joints is used to size many arrays in the system, and so it must be specified before any parameters that are array representations.

Error messages printed out by ARTFORMS will, in most cases, tell in which source file and in which line number within the source file the error occurred.

The parameters are described below with an example value that is either a default, or a reasonable value.

acura = 1.000000 ; The radius of the target; how close the hand must be to define a successful path segment.

agc_threshold = 10 ; This integer tells how large a value per coordinate must be so that the summed absolute value of the response recalled from the CMAC will be deemed by the stepsize control critic (AGC) to be a valid stored command. The example shows that for a 3 jointed arm, a summed value must be at least 3*10=30 to be accepted.

cmac_id = 0 ; An integer telling the id # of the direct inverse CMAC. An id # is 1-7, since up to 7 distinct CMACs may be allocated. A 0 indicates that the program should allocate a new one. If a non-zero is specified, the user should be certain that there really is a pre-existing CMAC allocated for that id #.

cmac_name = example ; This char * variable tells what the first name of a file used into which a CMAC image will be saved. The example indicates that cmac_id's CMAC will be saved in EXAMPLE.CMC, the repeller cmac will be saved in EXAMPLE.REP, etc.

constrainer_id = 0 ; An integer telling the id # of the K-CMAC.

constraints = 5 ; Which postural constraints to apply. 5 is the normal curvate constraint, 0 is no constraint, and other constraints weren't very successful.

dontask = 1 ; This integer flag if set to 1 suppresse the initial setup questions in the early part of the programs, and accepts defaults dictated by arm.ini. This is helpful for batch runs.

dontinhibit = 1 ; This integer flag = 1 means not to implement the ARTFORMS-1 style distributed plasticity CMAC (inhibitor_id).

dontnormalize = 0 ; This integer flag if set to 1 disables the normalization of steps (and input deltax, deltay vectors) to be unit vectors. Setting this to one is endpoint control.

dontrepel = 1 ; This integer flag if set to 1 means don't implement the ARTFOMRS-1 style reinforcement CMAC (repeller_id).

dontretrain = 0 ; 1 = don't retrain to a near optimal path, if after the CMAC contains information at a given state space point.

dont_show = 1 ; 1 means don't print out debugging information.

draw_at_target = 0 ; an int variable that controls whether or not a PostScript file showing the target postures will be printed out to a file called DATA.EPS. Value of 1 allows the output.

eta = 1 ; an int value giving the learning rate. 0 means 1.0, 1 means 1/2, 3 means 1/8, etc., for successive powers of 1/2.

general = 64 ; generalization parameter: the number of cells in a receptive field.

heuristic_mode = 6 ; Which kind of a priori heuristic to use to "guess" a move. 6 is the Berkinblitt synergy, 0 is random activation and the others weren't very successful.

inhibitor_id = 2 ; An integer telling the id # of the distributed plasticity CMAC (the inhibitor) CMAC.

joints = 900 900 900 ; An integer array containing the initial values of the joints as scaled integers with LSB representing 1/10 degree.

joint_scale = 10.000000 ; the joint angle increments, are scaled this much finer than the angles themselves.

kv = 1 1 1 ; A double precision array giving the initial value of the "curvate constraint" vector, K. The example is a normal curvate condition for a 4 jointed arm with equal angles.

lambda = 0.5 ; This double precision real value defines the learning rate (stepsize) for the postural constraint gradient training.

linkages = 10 12 15 ; This double precision real value defines the lengths of the articulator links.

max_joints 2100 2100 2100 ; This integer array gives the maximum permissible value (joint stops) for the joints. The example is for a 3 jointed arm, with upper joint stops at 210 degrees each.

max_passes = 1000 ; How many epochs (path segments) to execute.

max_step = 200 ; A scaled integer value that tells what the maximum magnitude (per coordinate) the heuristic generator will suggest. The example is 2.0 degrees.

memsize = 20000; The total number of physical memory vectors in the CMAC corresponding to cmad_id.

min_joints = 100 100 100 ; Like max_joints, but specifies the lower value joint stops.

nodot = 1 ; 1 = suppress the dot product heuristic.

noise = 0 ; This double precision floating point value specifies the heating value ($\sigma^2$) of the uniform white noise that will be injected into the critics' output signals.

no_adaptive_critic = 0 ; This integer flag if set to 1 disables the training and adaptive critics. This mode is acceptable for redundant sysems. If non-redundant, this will cause too little goal directedness, and the experiment will fail. It IS appropriate to set this to 1 for playback training, however.

num_joints = 3 ; Number of joints; for a value of 2, the arm is not redundant. This variable must precede specification in the ARM.INI file of joints, linkages, kv, max_joints, min_joints and quantization.

pause_arm = 0 ; 0 = starts with a live arm. 1 = starts with the arm paused. The space bar toggles this value.

phi_0 = 1.0 ; See the text under heuristic constraints.

phi_90 = -1.0 ; phi_0 and phi_180 are used to compute the inner and outer limits of the "allowed cone" for the heuristic and training critics.

playback = 0 ; See the variable "record", below.

practices = 1 ; This integer tells how many times each path segment is executed before going on the the next.

quantization = 4 4 8 8 8 ; this is an integer array telling how coarsely coded the input vector is. These values are read into the qnt_vec[] array. The example specifies a quantization of 4 for hand space and 8 for the joint space, for a 3 jointed arm.

ratchet = 1 ; If this int is 1 the ratchet condition is applied during training.

real_robot = 0 ; 1 = a SCORBOT is actually attached to COM port 2 0 = simulation ONLY.

record = 0 ; If this integer is 1, the trajectory steps will be recorded in a data file for playback later when the playback flag = 1.

repeller_id = 3 ; An integer telling the id # of the reinforcement CMAC for ARTFORMS-1. .

restart_mode = 1 ; This int flag determines if sequential reaching (0) or chained (1) trajectories will be formed.

retina_id = 3 ; An integer telling the id # of the R-CMAC.

scale_factor = 8 ; Scale factor for displaying the arm within screen limits.

script_file = first.ext ; This tells the program where to read a record of keystrokes from a script file. If the file doesn't exist, the program assumes that you want to create a new one. If it does exist, the program opens it and reads it in as if it were console input. This allows you to archive an experimental run for later display, perhaps with parameter changes.

smooth = 0 ; This int flag only applies when real_robot=1. 1 = don't execute the SCORBOT path until a full trajectory has been computed. This results in a smooth, fast arm trajectory, but reinforcement is impossible, and the simulator and arm are not synchronized. 0 = execute incremental moves of the SCORBOT arm as each step is computed. This results in a slow, jerky trajectory.

sounds = 0 ; An int flag that enables (1) or disables (0) the output of an auditory signal when a critic detects a failure.

tapered = r ; A value of 'r' means rectangular receptive fields, 'l' means linear tapered, 'o' means Albus style rectangular field.

target_file_name = NEW.TGT ; A char * variable pointing to the name of the file that has the target coordinates in it.

target_mode = 0 ; An integer variable. If zero it means read all targets in at once. If non-zero, it means: start out by reading in one target, and then after every target_mode path segments, read in another target position until all targets are read in.

teacher = 0 ; An int flag that tells what training mode to use. The normal mode is 0 which is default inverse modeling (train at the observed context) mode. Mode 1 trains at the desired context. Mode 2 trains at both contexts, mode 3 is used for endpoint control.

xtgt = 36.350000 ; The target, which appears as a small red circle.

ytgt = 25.450000 ; Appears at the x,y coordinates given. These coords are overwritten when the target file is read in.

## *Part III: The Critics.*

The functions, HeuristicCritic() and AdaptiveCritic(), apply the heuristic constraint rule. They determine to what extent an arm move must locally reduce the hand to target distance.

In simple terms, the procedure involves computing the dot product of the hand-to-target vector and the hand to new position vector. If the angle formed by the two vectors is 0 then the putative move would follow a tight rectilinear trajectory. Left or right of the straight line path, the two angles, phi_0, and phi_180, define the constraints. The new trajectory step MUST fall between phi_0 and phi_180. So a loose trajectory is allowed by, for instance, phi_0=0 and phi_90=180 degrees. a VERY tight trajectory is phi_0=phi_90=0, which actually is unachievable. If phi_0 is increased, it forces the trajectory away from a straight line (for obstacle avoidance), if phi_90 is small straight lines are favored. If it is increased towards 180 deg. curved trajectories are allowed. Caution: phi_0 and phi_90 are actually stored as cos(phi_0) and cos(phi_90), so they range from -1.0 to 1.0; So the loosest possible trajectory is phi_0=1.0 and phi_180=-1.0. phi_0=1.0 and phi_90=0 specify loosest constraint that doesn't allow the hand to target distance to increase.

## *Part IV Script Mode.*

If you specify a script file name in arm.ini, the program will use it for console input. If the file doesn't exist the program will prompt for input when necessary and save a new script file containing those prompted inputs as well as any asynchronous inputs supplied by the user through the keyboard. There are some problems related to WHEN you type input into the program which are described below.

Some idosyncrasies of script mode:

The keystrokes that are typed in response to prompted answers to questions are no problem. If they are keystrokes that are typed asynchronously while the program is executing, the program attempts to insert "wait-states" in the script file that will cause the Getch() function to return nothing until the number of wait states that were observed and recorded during creation of the script file have elapsed during execution of the program under script file control. The script files are also editable. This gives rise to problems related to control chars, function keys, etc.

If the program is reading console input from a script file, when "special keys" are read in, two character key sequences are read in, where the first is an ASCII null. The null can't be written to a script file, so the key '|' is substituted. Thus, in a script file, an UP ARROW looks like:

```
|

P
```

During execution of Yes(), if an actual keystroke is sensed, if it is 'Q' or 'q', the program terminates. If, on the other hand, a pNNNN is in the script file where Y or N is expected, a concert A (440 Hz) is sounded and NNNN milliseconds of delay occurs. This allows a demo to freeze a final result for a period of time.

Getch() stops and reads a keystroke delimited by CR LF. Otherwise it emulates getch(); It does however leave the rest of the line in the buffer getch_buf[];

get_ch() reads a single key from the console, if no script. If a script is being read, signified by the file RDF being active (read data file) Getch() is called to read a keystroke in from the script file. If a script is being written a single keystroke is read from the console and written to the script prefixed by a line reading \nnnn where nnnn is an integer representing the current kb_counter, which counts how many times KbHit() has been called to poll for a keystroke. Later, when a script is being read in, KbHit() watches for the value of kb_counter to equal a value of next_event, which it has read in from the script to tell it how many cycles into the program execution an asynchronous keystoke is expected. When reading a script line, if the first character is \, the program knows that this is a signal that an asynchronous event is coming up. Thus is is necessary for the whole system to "preread" the script file, while polling, so that if an asynchronous input is coming up it will know ahead of time. When it reads the input, then if the line was NOT prefixed by '\', the flag "getch_buf_full" is set to -1 to tell the system that next time a synchronous read (Getch()) occurs, the data is already in the buffer.

## *Part V: Learning evaluation and reinforcement.*

The arm learns very quickly to move about in the workspace. In a sense, this is a form of obstacle avoidance, in that the robot learns to modify the inverse kinematic model to account for joint stops.

As learning progresses, the planner gets "smarter", a local attribute that is dependent on the arm position. This property is shown on the screen by the color of the arm and hand. When the CMAC is untrained, the arm shows up in grey, followed by white, green, yellow, red and finally blue in sequence as the degree of training of the CMAC increases in a state space region.

So, as the arm sweeps out a trajectory, one would expect its behavior to be more erratic when displayed in white and more smooth when in blue. In fact, when in blue, the arm can't learn at all... it is frozen with its current weights. But again, this is only local. The arm may be blue over part of its workspace, and white over other parts, which it hasn't visited yet.

Depressing alt-O during operation tells the arm that it has encountered an "obstacle". At that time, it tries to retrain with a bias against what it "used to think" it should do in the neighborhood of the obstacle, and the level of experience in that region is reduced back to novice level, to ensure that it can retrain. The presence of the bias against current knowledge should cause the old trajectory formation to move about to accommodate the obstacle. Subsequent trajectories in this neighborhood should detour around the obstacle.

## *Part VI: Constraint satisfaction.*

This subject is exhaustively covered in Chapter IV of the text, so no treatment is given here.

## *Part VII: HEURISTIC ERRORS*

In the code that tests for target directedness (subroutines Critic(), HeuristicCritic() and AdaptiveCritic(); see file disredux.c) there are numerous ways in which an error can be reported.

The following return codes are reported by system that mean, "there is an error reported by a critic". The source of the error by return code is:

- 1 = Sum of delta joint angles = 0
- 2 = sum of delta hand moves = 0
- 3 = joint space dot product heuristic failed (TryToLearn())
- 4 = heuristic mode 4 or 5 failed to have ascending magnitudes
- 5 = a zero length heuristic step was proposed
- 6 = the heuristic criterion failed (move outside the "allowed cone")
- 7 = HeuristicCritic() failed
- 8 = failure in HeuristicStep() after 5 tries to suggest a unit length step
- 9 = no data in the CMAC; virgin terrain!
- 10 = delta joint move vanished in AdaptiveStep() due to normalization
- 11 AdaptiveCritic failed
- 12 Numerical problem normalizing vector in HeuristicCritic()

## *Part VIII: SELECTED SUBROUTINES EXCERPTED FROM THE SOURCE CODE*

```
// From the file ARTISTS.C:
int inp_vec[NUM_INPS];
int delta_joint[NUM_JOINTS],dot;

// From the file CONSTRAN.C:
int K[NUM_JOINTS+1];
int bump=0;
static int i;
void GetKVector(void)
{
     if (!rembr(constrainer_id, inp_vec, K))
        TOOBAD("constrainer CMAC can't rembr");
     LOOP(i,num_joints-1)
     {  K[i] += 100;
     }
}

void TrainKVector(void)
{
// K is scaled so 100 = 1.0.
//    Remove the excess so a null CMAC represents
    // a K vector that is (1,1,1,1...)
    LOOP(i,num_joints-1)
    {   K[i] -=  100;
    }
    learn(constrainer_id,inp_vec, K, eta);
}

void AdjustKVector(int resp, int joint)
{   double gain = (double) resp/100.0;
    GetKVector();
    if (joint1)
    {  K[joint-1] += (int)(10*gain);
    }  K[joint]   -= (int)(10*gain);
       K[joint+1] += (int)(10*gain);
    TrainKVector();
}


// From the file, TRY2LERN.C:

double ddj[NUM_JOINTS];
double kv[NUM_JOINTS-1];

void ddjUpdate(void)
{  int i;
  LOOP(i,num_joints)
  {  delta_joint[i]+=ddj[i];
  }
}

static double f(int i)
{   return kv[i-1] * (double) inp_vec[i+2]
          -  kv[i] * (double) inp_vec[i+3];
}

void SatisfyPosturalConstraints(int flag)
{ int i;
  double f0,f1;
  i=i;
  LOOP(i,num_joints)
  {  ddj[i]=0.0;
  }
  for(i=1;i < num_joints;i++)
  { if (i > 1)f0=f(i-1);
    f1=f(i);
    if (i==1)
    {  ddj[i] = -lambda*kv[i-1]*f1;
    }
    else if (i==num_joints-1)
    {
       ddj[i] = lambda*kv[i-1]*f0;
    }
    else
    {
       ddj[i] = lambda*kv[i-1]*(f0-f1);
    }
  }
}
```

```
int obstacles = 0;
int TryToLearn(void)
{
  int i,dot,OK;
  if ((constraints==1 || constraints== 4|| constraints== 5) && obstacles)
  {  GetKVector();
     // Put the K vector that is derived from the R-CMAC and K-CMAC into
     // the working array k[ ]
     LOOP(i,num_joints-1)
        { kv[i] = (double) K[i] / 100.0;}
  }
  if (K[0]+K[1] != 200) i=i;

// This prevents a zero length joint move being computed

   OK=0;
   LOOP(i,num_joints)
   {  if(delta_joint[i])
      {  OK=1;
        break;
      }
   }
   if (!OK) return 1;
   if (inp_vec[0]==0 && inp_vec[1]==0) return 2;

// train the CMAC memory for the direct inverse modeling training step
       learn(cmac_id, inp_vec, delta_joint,eta);
       if (constraints)
       {  if (!CloseEnough())
          {
            LOOP(i,num_joints)
            {  ddj[i]=0.0;
            }
            // apply the postural constraint
            rembr(cmac_id, inp_vec, delta_joint);
            switch(constraints)
            { case 5:  // The other constraints aren't listed here.
                SatisfyPosturalConstraints(0);
                ddjUpdate();
                break;

               default: printf("\nDon't know what constraint %d is",
                               constraints);
               TheEnd(1);
             }
           // train the CMAC memory for the postural constraint
           learn(cmac_id, inp_vec, delta_joint, eta);
          }  // End of CloseEnough() conditional
          else
          {  fprintf(inf_file, "CLOSE ENOUGH PROBLEM in file %s, line %d,
             pass# %d", FILE,__LINE, PassNumber );
          }
        }

   return 0;
}
```

# *BIBLIOGRAPHY*

**Adler ['67]** Adler, C.F., *Modern Geometry, 2d Edition*, McGraw-Hill, 1967.

**Albus ['79a]** Albus, James S., "Methods of Planning and Problem Solving In the Brain", Mathematical Biosciences 45:247-293, 1979.

**Albus ['79b]** Albus, James S., "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC), Transactions of the ASME, Sept. 1979.

**Albus ['81]** Albus, James S., *Brains, Behavior and Robotics*, McGraw Hill, 1981.

**Almeida ['89]** Almeida, Luis, "Back Propagation in Non Feedforward Networks", from *Neural Computing Architectures*, MIT Press, 1989.

**An ['91]** An, P., *An Improved Multidimensional CMAC Neural Network: Receptive Field Function and Placement*. A doctoral dissertation presented to the University of New Hampshire ECE Dept., 1991.

**Anderson ['88]** Anderson, James A., Arthur B. Markman, Susan R. Viscuso and Edward J Wisniewski, "Programming Neural Networks", Neural Networks, vol. 1, supplement 1, p. 157. Sept. 1988.

**Andrews ['83]** Andrews, J. R. and Hogan, N. "Impedance control as a framework for implementing obstacle avoidance in a manipulator". In D. E. Hardt and W.J. Book, eds., *Control of Manufacturing Processes and Robotic Systems*. New York:American Society of Mechanical Engineers, 1983.

**Atkeson ['88]** Atkeson, C.G., and David J. Reinkensmeyer, "Using Associative Content-Addressable Memories to Control Robots", MIT AI Lab., Tech. Report NE43-759, 1988.

**Barto ['83]** Barto, A. G., Richard S. Sutton and Charles W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Problems", IEEE Transactions on Systems, Man and Cybemetics, Vol. SMC-13, No. 5, Sept./Oct., 1983.

**Barto ['89]** Barto, A. G., "Connectionist Learning for Control: An Overview", COINS Technical Report 89-89, University of Mass., Amherst, MA, Sept. 1989.

**Block ['62a]** Block, H.D.,"The Perceptron: A Model for Brain Functioning", Review of Modern Physics, Vol. 34 No. 1, Jan. 1962.

**Berkinblitt ['90]** Berkinblitt, I.M., Gelfand, J., and Feldman, "Model of the Control of the Movements of a Multijoint Limb", *Biophysics*, vol. 31, no. 1, 1986, pp. 142-153.

**Block ['62b]** Block, H.D., B.W. Knight, Jr., and F. Rosenblatt, "Analysis of a Four-Layer Series coupled Perceptron.", Reviews of Modern Physics, Vol. 34, Number 1, Jan. 1962.

**Brooks ['89a]** Brooks, Rodney A. "Achieving Artificial Intelligence Through Building Robots", MIT AI Lab, Tech. A.I. Memo 899.

**Brooks ['89b]** Brooks, Rodney A. "Planning is Just a Way of Avoiding Figuring Out What to Do Next", MIT AI Lab, A.I. Working Paper 303.

**Brooks ['88]** Brooks, Rodney A. "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network", Mit AI Lab., Sept. 1988.

**Brooks ['91]** Brooks, Rodney A. "Intelligence Without Reason", IJCAI, 1991.

**Brooks ['91a]** Brooks, Rodney A. "Intelligence without representation", Artifical Intelligence, Elsevier Science Publishers B.V., (47), 1991, pages 139-159.

**Bullock ['88]** Bullock, D., Grossberg, S., "Neural Dynamics of Planned Arm Movements. Emergent Invariants and Speed Accuracy Properties During Trajectory Formation", from *Neural Networks and Natural Intelligence*, Grossberg, S., Ed., MIT Press, 1988.

**Canney ['90b]** Canney, John, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

**Carter ['90]** Carter, M.J., Rudolph, F.J. and Nucci, A.J. 1990. "Operational Fault Tolerance of CMAC Networks"*Advances in Neural Information Processing Systems* 2. Touretsky, D.S., ed. San Mateo, CA: Morgan Kaufman.

**Carter ['90b]** Carter, M.J., Nucci, A.J., Miller, W.T. III, An, E., Rudolph, F.J., "Slow Learning Scenarios for Locally Generalizing Neural Networks and Implications for Fault Tolerance", *24th Annual Conference on Information Sciences and Systems*, March 21-23, 1990, Princeton, N.J.

**Carter ['91]** Carter, M.J., Nucci, A.J., An, E., Miller, W.T. and Rudolph, F.J. "Slow Learning in CMAC Networks and Implications for Fault Tolerance", (In preparation).

**Caudill ['90]** Caudill, Maureen, and Butler, Charles, *Naturally Intelligence Systems,* MIT Press 1990.

**Crick ['89]** Crick, Francis, "The recent excitement about neural networks", *Nature*, vol 337, no. 32, 12 Jan., 1987.

**Fahlman ['90]** Fahlman, Scott E., and Christian Lebiere, "The Cascade Correlation Learning Architecture", from *Advances in Neural Information Processing Systems* 2, Morgan Kaufman, 1990. (in press)

**Flash ['85]** Flash, T. and Hogan, N. "The coordination of arm movements: An experimentally confirmed mathematical model", Journal of Neuroscience, 5:1688-1703, 1985.

**Gillette ['86]** Gillette, Rhanor, "The Role Of Neural Command In Fixed Action Patterns Of Behaviour", *Aims & Methods - Neuroethology*, D.M. Guthrie, ed. Manchester University Press, pp. 46-79. 1986.

**Glanz ['87]** Glanz, F. H., Miller, W. T., "Shape Recognition Using a CMAC Based Learning System." Proceedings SPIE: Intelligent Robots and Computer Vision, Cambridge, Mass., Nov., 1987.

**Glanz ['89]** Glanz, F. H., and Miller, W. T., "Deconvolution and Nonlinear Inverse Filtering Using a Neural Network." Proc. ICASSP '89, Glasgow, Scotland, May 23-26, 1989, vol. 4, pp. 2349-2352.

**Grossberg ['88]** Grossberg, S., "Nonlinear Neural Networks,: Principles, Mechanisms, and Architectures", Neural Networks, Vol 1, pp. 17-61, 1988.

**Grossberg ['88]** Grossberg, Stephen, *Neural Networks and Natural Intelligence*, MIT Press, 1988.

**Handelman ['89]** Handelman, David A., Stephen H. Lane, and Jack J. Gelfand, "Integrating Knowledge-based System and Neural Network Techniques for Autonomous Learning Machines", International Joint Conference on Neural Networks, Wash. DC, June 1989.

**Handelman ['90]** Handelman, David A., Lane, Stephen H., "Integration of Knowledge-based Systems and Neural Networks for Intelligent Sensorimotor Control", Robicon Systems, Inc., RSI Tech., Report TR90-1001, Princeton, N.J., Oct. 1990.

**Herold ['88]** Herold, D. J., Miller, W. T., Kraft, L. G., and Glanz, F. H., "Pattern Recognition Using a CMAC Based Learning System." Proceedings SPIE: Automated Inspection and High Speed Vision Architectures II, vol. 1004, pp. 84-90, 1988.

**Hewes ['88a]** Hewes, R.P., *Implementation and Demonstration of a Learning Control Test System For a Five Axis Industrial Robot*, Masters Thesis, Dept. of Electrical and Computer Engineering, University of New Hampshire, May, 1988.

**Hewes ['88]** Hewes, R.P., and Miller, W.T. "Practical Demonstration of a Learning Control System for a Five Axis Industrial Robot." Proceedings SPIE: Intelligent Robots and Computer Vision, vol. 1002, 1988.

**Hinton ['84]** Hinton, Geoffrey, "Parallel Computations for controlling an Arm", Journal of Motor Behavior, 1984, Vol. 16 No. 2, 171-194.

**Hofstadter ['79]** Hofstadter, Douglas R., *Godel, Escher, Bach: An Eternal Golden Braid*, Basic Books, New York, 1979.

**Hogan ['80]** Hogan, N. "Mechanical Impedance Control in Assistive Devices and Manipulators", Proceedings of 1980 Joint ACC Conference, San Francisco, 1980; p TA10-B.

**Hogan ['84a]** Hogan, N. "Impedance Control: An Approach to Manipulation", *Proceedings of the 1984 American Controls Conference*, June 6-8, 1984; vol 1, pp 304-13.

**Hogan ['84]** Hogan, N. "An Organizing Principle for a Class of Voluntary Movements", Journal of Neuroscience, 4:2745-2754, 1984.

**Hogan ['85]** Hogan, N. "Impedance Control: An Approach to Manipulation", ASME Journal of Dynamic Systems, Measurement and Control, vol 107,#1: pp 1-24, March 1985.

**Hogan ['85]** Hogan, N. , "Adaptive Control of Mechanical Impedance by Coactivation of Antagonist Muscles", IEEE Transactions on Automatic Control, vol AC-29, #8, pp 681-90, Aug. 1984.

**Hogan ['92]** Hogan, N. , "How Humans Adapt to Kinematic Constraints", In: Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems, New Haven, CT, May 20-22, 1992, pp. 182-185

**Holmes ['91]** Holmes, E.F. and Carter, M.J. "Fault Tolerance During the Learning Stage in CMAC Networks." In preparation for submission to IEEE Trans. Neural Networks.

**Hornik ['89]** Hornik, K., Stinchcombe, M. and White, H.," Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, pp 359-366, 1989.

**Houk ['88]** Houk, James, C., "Schema For Motor Control Utilizing a Network Model of the Cerebellum", Neural information Processing Systems. D.Z. Anderson, ed. NY, NY: Amer. Inst. Physics, 1988:367-376.

**Houk ['89]** Houk, James, C., "Cooperative Control of Limb Movements by the Motor Cortex, Brainstem and Cerebellum", from *Models of Brain Function*, Ed. Rodney M.J. Cotterill, Cambridge University Press, 1989.

**Houk ['90]** Houk, James C., Singh, S.P., Fisher, C., Barto, A., "An Adaptive Sensorimotor Network Inspired by the Anatomy and Physiology of the Cerebellum", from Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control*, Cambridge MA, MIT Press, December, 1990.

**Houk ['91]** Houk, James C., "Outline for a Theory of Motor Learning", *Tutorials in Motor Neuroscience, Proceedings of the NATO ASI Corsica Meeting,* 16-24 September, 1990, G.E. Stelmach, ed., Kluwer Academic Publishers, 1991

**Houk ['92]** Houk, James C., "Learning In Modular Networks", In: Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems, New Haven, CT, May 20-22, 1992, pp. 80-84

**Hwang ['88]** Hwang, Y. K. and Ahuja, N. "Path planning using a potential field representation". Coordinated Science Laboratory Technical Report UILU-ENG-88-2251, University of Illinois, Urbana-Champaign. 1988.

**Jones ['87]** Jones, William P., and Josiah Hoskins, "Back-Propagation: A Generalized Delta Rule", BYTE, Oct. 1987. 155-162.

**Jordan ['88]** Jordan, Michael I., "Supervised learning and systems with excess degrees of freedom", U. Mass. Amherst, COINS Tech. Report 88-27, May 1988.

**Jordan ['90]** Jordan, Michael I., "Forward Models: Supervised learning with a distal teacher", MIT Center for Cognitive Science Occasional Paper #40, (submitted for publication to Cognitive Science, 1990).

**Josin ['87]** Josin, Gary, "Neural-Network Heuristics: Three Heuristic Algorithms That Learn From Experience", BYTE, Oct. 1987. 183-192.

**Kawato ['89]** Kawato, M., Y. Maeda, Y. Uno and R. Suzuki, "Trajectory Formation of Arm Movement by Cascade Neural Network Model Based on Minimum Torque-change Criterion", ATR Auditory and Visual Perception Research Laboratories internal paper, 7/26/89.

**Kawato ['89]** Kawato, M., Y., from Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control,* Chapter 9, Cambridge MA, MIT Press, December, 1990.

**Khatib ['85]** Khatib, O. (1985). "Real-time obstacle avoidance for manipulators and mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation,* St. Louis, MO, March.

**Khatib ['86]** Khatib, O. (1986). "Real-time obstacle avoidance for manipulators and mobile robots". International of Journal of Robotics Research, 5(1) pp 90-98.

**Klein ['83]** Klein, Charles, and Chiang-Hsiang Huang, "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators", IEEE Trans. on Sys. Man, and Cyber. vol. SMC-13, no. 3, Mar/Apr. 1983.

**Korein ['85]** Korein, James U., *A Geometric Investigation of Reach,* MIT Press, 1985.

**Kraft ['89a]** Kraft, L. G., and Campagna, D. P., "A Comparison of CMAC Neural Network and Traditional Adaptive Control Systems." Proc. of the 1989 American Controls Conf., Pittsburgh, Pa., May, 1989.

**Kraft ['89b]** Kraft, L. G., and Campagna, D. P., "Comparison of Convergence Properties of CMAC Neural Network and Traditional Adaptive Controllers." Proc. 28th Conf. on Decision and Control, Tampa, Fla., December, 1989, pp. 1744-1745.

**Kraft ['89]** Kraft, L. G., An, E., and Campagna, D. P., "Comparison of CMAC Controller Weight Update Laws." Proc. 28th Conf. on Decision and Control, Tampa, Fla., December, 1989, pp. 1746-1747.

**Kraft ['90a]** Kraft, L. G., Miller,W.T., Glanz, F.H., Prina, S., "A Neural Network Based Controller For a Magnetic Disc Drive With a Noisy Sensor", Procedings of Electronics Imaging 1990 East Conference, Boston, MA, October, 1990.

**Kraft ['90b]** Kraft, L. G., "Applications of CMAC to Optimal Control Problems", To appear in the Proceedings of the Workshop on Aerospace Applications of Neural Control, co-sponsored by the National Science Foundation, McDonnell Douglas, and Washington University, St. Louis, Missouri, October, 1990.

**Kraft ['90c]** Kraft, L. G., and Campagna, D.P., "Comparison of CMAC Architectures for Neural Network Control", IEEE Decision and Control Conference, Honolulu, Hawaii, December, 1990.

**Kraft ['90]** Kraft, L. G., and Campagna, D. P., "A Summary Comparison of CMAC Neural Network and Traditional Adaptive Control Systems." IEEE Control Systems Magazine, April, 1990.

**Kraft ['91]** Kraft, L. G., An, Edgar, and Briggs, E., "Convergence Properties of CMAC Neural Network Controllers", Submitted to the 1991 American Controls Conference, Boston, Mass.

**Kramer ['92]** Kramer, G., *Solving Geometric Constraint Systems*, MIT Press, Cambridge, MA, 1992.

**Kuffler ['84]** Kuffler, S.W., Nicholls, J.G., Martin, A.R., *From Neuron to Brain*, Sinauer, Sunderland, MA, 1984.

**Kuhn ['90]** Kuhn & Herzberg, "Variations on Training of Recurrent Networks", from 24th Conference on Information Sciences & Systems, Princeton, N.J. 3/21/90.

**Kuperstein ['88]** Kuperstein, Michael, and Jorge Rubinstein, "Implementation of an Adaptive Neural Controller for Sensory-Motor Coordination", IEEE Control Systems Magazine, April, 1989,on and lecture given at the Boston INNS conference, Sept. 1988).

**Lippmann ['87]** Lippmann, Richard P., " An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, Apr. 1987.

**Lozano-Leal ['89]** Lozano-Leal, R., "Robust Adaptive Regulation Without Persistent Excitation", *IEEE Transactions on Automatic Controls*, vol. 34, no. 12, pp. 1260-1267, June 1989.

**Lozano-Perez ['87]** Lozano-Perez, T., " A simple motion planning algorithm for general robot manipulators", *IEEE Journal of Robotics and Automation*, 1987, RA-3, pp 224-238.

**MacLean ['73]** MacLean, P.D.A *TriuneConcept of the Brain and Behavior*,Toronto: University of Toronto Press, 1973.

**Marr ['69]** Marr, D., "A Theory of Cerebellar Cortex", Journal of Physiology, London, Eng., V. 202, 1969, pp 437-470.

**Marshall ['87]** Marshall, S.V., Skitek, G.G., *Electromagnetic Concepts and Applications*, 2nd Edition, Prentice-Hall, 1987, Englewood Cliffs, New Jersey.

**Massone ['89]** Massone, L., Bizzi, E., "A Neural Network Model for Limb Trajectory Formation", *Biological Cybernetics*, 61, 417-425, 1989.

**McCulloch ['43]** McCulloch, W.W and Pitts, W., "A logical calculus of the ideas imminent in neurvous activity", *Bulletin of Mathematical Biophysics*, 5:115-33, 1943

**Mel ['87]** Mel, Bartlett W., "MURPHY: A robot that learns by doing", Proc. of the IEEE Conf. on Neural Information Processing Systems, Denver, 1987.

**Mel ['89]** Mel, Bartlett W., "MURPHY: A Neurally-Inspired Connectionist Approach to Learning and Performance in Vision-Based Robot Motion Planning.", Tech. Report CCSR-89-17A, Center for Complex Systems Research, Univ. of Ill., Urbana-Champaign.

**Mel ['90]** Mel, Bartlett W., "Vision-Based Robot Motion Planning", from Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control*, Cambridge MA, MIT Press, December, 1990.

**Miller ['78]** Miller, W.T., Geselowitz, D.B., "Simulation studies of the electrocardiogram. I. The normal heart." Circulation Research vol 43, pp 301-314, 1978.

**Miller ['78a]** Miller, W.T., Geselowitz, D.B., "Simulation studies of the electrocardiogram. II. Ischemia and infarction." Circulation Research vol. 43, pp. 315-329, 1978.

**Miller ['86]** Miller, W. T., "A Nonlinear Learning Controller for Robotic Manipulators." Proc. of the SPIE: Intelligent Robots and Computer Vision, vol 726, pp. 416-423, October, 1986.

**Miller ['87]** Miller, W. T., "A Learning Controller for Nonrepetitive Robotic Operations." Proc. of the Workshop on Space Telerobotics, JPL publication 87-13, vol. II, pp. 273-281, Pasadena, CA, January 19-22, 1987.

**Miller ['87]** Miller, W. T., "Sensor Based Control of Robotic Manipulators Using A General Learning Algorithm." IEEE J. of Robotics and Automation, vol. RA-3, pp. 157-165, April, 1987.

**Miller ['87]** Miller, W. T., Glanz, F. H., and Kraft, L. G., "Application of a General Learning Algorithm to the Control of Robotic Manipulators." The International Journal of Robotics Research, vol. 6.2, pp. 84-98, Summer, 1987.

**Miller ['88a]** Miller, W. T., "Real Time Application of Neural Networks for Sensor-Based Control of Robots with Vision", Internal paper of Univ. of N.H, ECE Dept., revised, Sept. 15, 1988

**Miller ['88b]** Miller, W. T., Hewes, R.P., Glanz, F.H., and Kraft, L.G., "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller", Journal of Robotics and Automation. (date and volume???)

**Miller ['88c]** Miller, W. T., "Real Time Learned Sensor Processing and Motor Control for a Robot with Vision." Proceedings of the First Annual Conference of the International Neural Network Society, Boston, MA, September 6-10, 1988, pp. 347.

**Miller ['88d]** Miller, W. T., and Hewes, R.P., "Real Time Experiments in Neural Network Based Learning Control During High Speed, Nonrepetitive Robot Operations." Proceedings of the Third IEEE International Symposium on Intelligent Control, Washington, D.C., August 24-26, 1988.

**Miller ['89]** Miller, W, T., "Real Time Application of Neural Networks for Sensor-Based Control of Robots with Vision." IEEE Transactions on Systems, Man, and Cybernetics Special Issue on Information Technology for Sensory-Based Robot Manipulators, Vol. 19, pp. 825-831, August, 1989.

**Miller ['90a]** Miller, W. T., and Aldrich, C. M., "Rapid Learning Using CMAC Neural Networks: Real Time Control of an Unstable System." Proceedings of the Fifth IEEE International Symposium on Intelligent Control, Phil., PA, Sept. 5-7, 1990, pp. 465-470.

**Miller ['90b]** Miller, W. T., Box, B. A., and Whitney, E. C., "Design and Implementation of a High Speed CMAC Neural Network Using Programmable CMOS Logic Cell Arrays." UNH Intelligent Structures Group Report ECE.IS.90.01, Univ. of New Hampshire, Feb. 6, 1990.

**Miller ['90c]** Miller, W. T., Box, B. A., Whitney, E. C., and Glynn, J. M., "Design and Implementation of a High Speed CMAC Neural Network Using Logic Programmable CMOS Logic Cell Arrays", Proceedings of the IEEE Conference on Neural Information Processing Systems, Denver, CO,Nov. 26-29, 1990.

**Miller ['90d]** Miller, W. T., Glanz, F. H., and Kraft, L. G., "CMAC: An Associative Neural Network Alternative to Backpropagation." IEEE Proceedings, Special Issue on Neural Networks II: Analysis, Techniques, and Applications, vol. 78, pp. 1561-1567, October, 1990.

**Miller ['90e]** Miller, W. T., Hewes, R. P., Glanz, F. H., and Kraft, L. G., "Real Time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller." IEEE J. of Robotics and Automation, Vol. 6, pp. 1-9, 1990.

**Miller ['90f]** Miller, W. T., Latham, P. J., and Scalera, S. M., "Bipedal Gait Adaptation for Walking with Dynamic Balance: Temporal Difference Learning Using CMAC Neural Networks." Presented at the Conference on the Simulation of Adaptive Behavior: From Animals to Animats, Paris, France, September 24-28, 1990.

**Miller ['90g]** Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control*, Cambridge MA, MIT Press, December, 1990.

**Miller ['90]** Miller, W. T., An, E., Glanz, F. H., and Carter, M. J., "The Design of CMAC Neural Networks for Control." Proceedings of the Sixth Yale Workshop on Adaptive Systems, New Haven, CT, August 15-17, 1990, pp. 140-145.

**Miller ['91]** Miller, W. T., Latham, P. J., and Scalera, S. M., "Bipedal Gait Adaptation for Walking with Dynamic Balance." Submitted to the 1991 American Controls Conference, Boston, Mass.

**Minsky ['69]** Minsky, Marvin, and Seymour Papert, *Perceptrons, an Introduction to Computational Geometry*, MIT Press, 1969.

**Moody ['89]** Moody, John, and Christian J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units", Neural Computation, 1989.

**Mozer ['89]** Mozer, Michael, "Discovering the Structure of a Reactive Environment By Exploration", from*Advances in Neural Information Processing Systems* 2, Morgan Kaufman, 1990.

**Newman ['85]** Newman, W. S. and Hogan, N. "High speed robot control and obstacle avoidance using dynamic potential functions", Proceedings of the IEEE International Conference on Robotics Information, St. Louis, MO, March 1985.

**Nguyen ['89]** Nguyen, Derrick, and Bernard Widrow, "The Truck Backer-Upper: an Example of Self-learning in NNs", International Joint Conference on Neural Networks, June, 1989.

**Omohundro ['87]** Omohundro, S. "Efficient algorithms with neural network behavior". Journal of Complex Systems, 1987, 1(2): 273-347.

**Noble ['67]** Noble, Ben, *Application of Undergraduate Mathematics in Engineering*, Macmillan, N.Y., 1967.

**Pao ['92]** Pao, Y., Takefuji, Y., "Functional-Link Computing: Theory, System Architecture, and Functionalities", *IEEE Computer*, May, 1992, vol. 25, no. 5, p. 76.

**Penrose ['89]** Penrose, Roger, *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, Oxford University Press, 1989.

**Pitts ['47]** Pitts, W. and McCulloch, W.W , "How we know universals", .*Bulletin of Mathematical Biophysics*, 9:127-47, 1947.

**Rivest ['87]** Rivest, Ronald L., and Robert E. Shapire, "A New Approach to Unsupervised Learning in Deterministic Environments", MIT Laboratory for Computer Science, Tech Report, 1986.

**Rudolph ['90]** Rudolph, Frank, "Locally Optimizing Neural Networks In Adaptive Robot Path Planning", International Joint Conference on Neural Networks, Jan. 16, 1990, Washington, D.C.

**Rumelhart ['86]** Rumelhart, D.E., J.L. McClelland, *Parallel Distrubuted Processing: Explorations in the Microstructure of Cognition* , MIT Press, 1986.

**Rumelhart ['88]** Rumelhart, D.E., J.L. McClelland, *Explorations in Parallel Distributed Processing*, MIT Press, 1988.

**Seegee ['92]** Segee, B., *Characterizing and Improving the Fault Tolerance of Neural Networks.* A doctoral dissertation presented to the University of New Hampshire ECE Dept., 1992.

**Selverston ['88]** Selverston, Allen I, "A Consideration of Invertebrate Central Pattern Generators as Computational Databases", Neural Networks, vol. 1 no. 2, pp. 109-117, 1988. Pergamon Press, New York.

**Sinkjaer ['90]** Sinkjaer, T., Wu, C.H. Barto, A., Houk, J. C., "Cerebellar Control of Endpoint Position -A Simulation Model", Proceedings of the International Joint Conference on Neural Networks, San Diego, Vol. II: 705-710, 1990.

**Shepansky ['87]** Shepanski, J.F., and S.A. Macy, Teaching Artificial Neural Systems to Drive: Manual Training Techniques for Autonomous Systems", SPIE Vol. 848, Intelligent Robots and Computer Vision: Sixth in a Series(1987).

**Snyder ['85]** Snyder, Wesley E., Industrial Robots: *Computer Interfacing and Control,* Prentiss Hall, 1985.

**Sutton ['90]** Sutton, R.S., "First Results with Dyna, An Integrated Architecture for Learning, Planning and Reacting", from Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control,* Chapter 8, Cambridge MA, MIT Press, December, 1990.

**Van de Vegte ['86]** Van de Vegte, John, *Feedback Control System.*, Prentice Hall, 1986.

**Wasserman ['88]** Wasserman, P.D., *Neural Computing: Theory and Practice,* Van Nostrand Reinhold, 1989.

**Wavering ['88]** Wavering, Albert J., "Manipulator Primitive Level Task Decomposition", NIST Technical Note 1256, U.S. Dept. of Commerce, NIST, Oct. 1988.

**Werbos, P. J. ['90]** Werbos, P.J. "A Menu of Designs for Reinforcement Learning", from Miller, W. T., Sutton, R. S., and Werbos, P. J., (editors), *Neural Networks for Control,* Cambridge MA, MIT Press, December, 1990.

**Williams ['89]** Williams, Ronald J., "Adaptive State Representation and Estimation Using Recurrent Connectionist Networks", College of Computer Science, Northeastern University internal paper.

# *INDEX*

Index entries are "**chapter**.page"; (the chapter numbers are in bold face type).