APPLIED INTELLIGENCE:

The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies

Volume 17, Number 1, July/August 2002

Introduction to the Special Issue: Scalable Applications of Neural Networks to Robotics ..... *Patrick van der Smagt and Daniel Bullock* 7

Searching a Scalable Approach to Cerebellar Based Control..... *Jan Peters and Patrick van der Smagt* 11

A Scalable Model of Cerebellar Adaptive Timing and Sequencing: The Recurrent Slide and Latch (RSL) Model ..... *Bradley J. Rhodes and Daniel Bullock* 35

Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning ..... *Stefan Schaal, Christopher G. Atkeson and Sethu Vijayakumar* 49

Modelling of Complete Robot Dynamics Based on a Multi-Dimensional, RBF-like Neural Architecture .... *Markus Krabbes and Chistian Döschner* 61

A Scalable Intelligent Takeoff Controller for a Simulated Running Jointed Leg ..... *Peggy Israel Doerschuk, Vinh Nguyen and Andrew Li* 75

Evolving Receptive-Field Controllers for Mobile Robots ..... *Ralf Salomon* 87

A Self-Organizing Context-Based Approach to the Tracking of Multiple Robot Trajectories ..... *Aluizio F.R. Araújo and Guilherme de A. Barreto* 99

Applied Intelligence 17, 99–116, 2002

A Self-Organizing Context-Based Approach to the Tracking of Multiple Robot Trajectories

ALUIZIO F.R. ARAÚJO AND GUILHERME DE A. BARRETO
Department of Electrical Engineering, University of São Paulo, São Carlos, Brazil
aluizioa@sel.eesc.sc.usp.br
gbarreto@sel.eesc.sc.usp.br

Abstract. We have combined competitive and Hebbian learning in a neural network designed to learn and recall complex spatiotemporal sequences. In such sequences, a particular item may occur more than once or the sequence may share states with another sequence. Processing of repeated/shared states is a hard problem that occurs very often in the domain of robotics. The proposed model consists of two groups of synaptic weights: competitive interlayer and Hebbian intralayer connections, which are responsible for encoding respectively the spatial and temporal features of the input sequence. Three additional mechanisms allow the network to deal with shared states: context units, neurons disabled from learning, and redundancy used to encode sequence states. The network operates by determining the current and the next state of the learned sequences. The model is simulated over various sets of robot trajectories in order to evaluate its storage and retrieval abilities; its sequence sampling effects; its robustness to noise and its tolerance to fault.

Keywords: robotics, trajectory tracking, neural networks, unsupervised learning, temporal context

1. Introduction

Robot learning presents a number of challenging problems, namely (1) they tightly integrate perception, decision making and execution; (2) robotic domains are usually complex, yet the expense of using actual robotic hardware often prohibits the collection of large amounts of training data; and (3) most robotic systems are real-time systems, implying that decisions must be made within critical or practical time constraints. Since other important real-world application domains share those characteristics, robotics is a highly attractive area for research on machine learning, especially within the field of artificial neural networks (ANNs).

The research in ANNs and its application in distinct domains makes it possible to investigate solutions to complex problems in robotics following different learning paradigms [1, 2]. A common problem in robotics is *trajectory tracking*, in which a robot is required to follow accurately a continuous path [3]. Such a task is mainly programmed by means of the so-called *walk-through* method in which an operator guides the robot through a sequence of desired arm positions. These positions are then stored in the controller memory for later recall. Such a method is time consuming and uneconomical, since during the walk-through process the robot is not engaged in productive activity [4] and the process is realized under complete supervision of the robot operator.

Tracking can easily be handled within the framework of artificial neural networks in which trajectories can be seen as a succession of arm configurations, i.e., a temporal sequence of arm positions, hence, neural models can model this type of processing. In particular, the unsupervised learning paradigm has appealing characteristics for its use in Robotics and temporal sequence processing. In unsupervised neural networks, behavior emerges by means of a self-organization process, thus reducing substantially the robot programming burden that accounts for as much as a third of the total cost of an industrial robot system [5]. Also, unsupervised models are often fast, encouraging their use in incremental

and on-line learning. Moreover, the structure of neural networks allows massive parallel processing [4] which enables the network to respond quickly in generating real-time control actions.

An important issue, usually not addressed in simulations and tests reported by the neural network literature, is the learning of multiple robot trajectories [4]. In some industrial operations, a robot is often required to perform more than one task. Hence, the robot controller must be able to track more than one trajectory. One of the goals of the present work is to develop an unsupervised learning neural network model to learn and retrieve multiple trajectories.

We have grouped the various neural models for unsupervised-temporal-sequence based robot control into three classes according to their approach to trajectory processing: (i) learning of perception-action trajectories, (ii) learning of robot trajectories, and (iii) formation of robot trajectories.

The first approach involves direct association between sensory data and desired actions [6]. This approach is used when a mobile robot is required to explore the world to build a model of it. As the robot navigates, it experiences a long sequence of perception-action pairs. As the storage of such a sequence is often not feasible, the researchers introduced mechanisms of sequence chunking and linking [7, 8]: a long sequence is broken into subsequences (chunks) which are stored by the robot and properly concatenated (linked) when their combination leads to reach a particular goal.

Two examples of this approach are the models proposed by Denham and McCabe [8] and Heikkonen and Koikkalainen [5]. Both systems were applied to autonomous robot navigation tasks in which the agent had to build a model of the world by exploration. Denham and McCabe employed a reward system to determine whether the learning of a sequence was based on the achievement of a goal or the detection of novelty. This system was implemented by using the unsupervised model proposed by Wang and Arbib [7]. Heikkonen and Koikkalainen introduced several control algorithms based on Kohonen Self-Organizing Map [9]. The knowledge was acquired from existing sequences as well as from the robot exploratory navigation. The authors simulated a robot that quickly learned to select suitable actions for a range of sensory situations, adapted nicely to changes in the environment, and collided less and less frequently as time went by. However, this approach is not stable against deviations of the trajectory. If the robotic system finds itself in an untrained position, off any learned trajectory, no appropriate control action may be produced [6].

In the second approach, a neural network must learn to associate consecutive states of a trajectory and store these transitions for total or partial reproduction of the memorized trajectory. Usually, for the purpose of recall, the network receives as input the current state of the robot and responds with the next state, to execute a pre-defined task. This approach has been applied to point-to-point trajectory control and trajectory tracking [10].

Althöfer and Bugmann [11] described a neural implementation of a resistive grid used to plan the path of a robot arm. This model has limitations such as jerkiness of the movements and an inaccurate final end-effector position due to the resolution constraints of grid-based methods. In the context of mobile robotics, Bugmann et al. [6] proposed a neural network which uses normalized RBF neurons to encode the sequence of states forming the trajectory of an autonomous wheelchair. The network operates by producing the next spatial position and orientation for the wheelchair. As the trajectory may pass several times over a particular point, phase information is added to the position information to avoid the aliasing problem [12]. This problem occurs when identical sensory inputs may require different actions from an autonomous system, depending on the context. The use of normalized RBF neurons creates an attraction field over the whole state space and enables the wheelchair to recover from perturbations.

The third approach entails the creation of a robot trajectory given only its initial and final (target) positions. The robot receives sensory information from the workspace and autonomously constructs some kind of inverse mapping. Typical examples of this approach are the works of Grossberg and Kuperstein [13], Kuperstein and Rubinstein [14], Martinez et al. [15], and Ritter et al. [16]. The three first works describe a self-organizing model for visuomotor coordination of a robot arm. This model learns to control a 5-degree-of-freedom (DOF) robot arm to reach cylindrical objects. The authors use a set of one-dimensional topographic maps that represent the location of the target object and whose adaptive weights determine the output to the arm actuators. Each one-dimensional map has a fixed topographic ordering and only the output weights can be adapted during the learning process. As a consequence, the range of the expected input values must be known in advance and adaptive changes in the resolution of the neural population required for control are not possible.

Furthermore, as the maps are one-dimensional and their outputs are a linear summation for each actuator, they can approximate only a restricted class of control laws. The work of Bullock and Grossberg [17] extends Kuperstein's model by including muscle dynamics, initial conditions, muscle contraction rates, and feedback signals from muscle sensors.

Martinez et al. [15] and Ritter et al. [16] presented an approach to diminish the drawbacks of the Kuperstein model, by using 3D variant of the Kohonen SOM. In this approach the ordering and resolution of the map evolve during learning (by updating a layer of input weights), thus determining the distribution of the neural units over the task space, and overcoming the problem of fixed resolution of Kuperstein's model. The adaptation of the output weights was achieved by an error-correction procedure based on the Widrow-Hoff learning rule for adaptive linear elements. The 3D map eliminates restrictions arising from the additive coupling of several 1D maps and allows many neighboring units to cooperate during learning, increasing the efficiency and robustness of the algorithm. The authors reported simulation results in which after 30,000 training steps there are no significant positioning errors. This model was implemented in a 560 PUMA robot [18], producing small positioning errors. Moreover, the system was able to adapt to sudden changes of its geometrical parameters.

Despite the appealing features of the unsupervised learning-based control system, its use has been limited to a few model proposals, in part because a major part of the work on this topic is devoted to other paradigms such as supervised and reinforcement learning. In this paper we emphasize the feasibility of applying unsupervised learning to complex robotics problems.

We are particularly concerned with the problem of fast and accurate learning of single and multiple sequential patterns that represent robot trajectories. An unsupervised neural network algorithm is the chosen learning strategy mainly because it is based on self-organization. This principle has proved to be a rather generic technique to be employed in a wide range of application domains, such as robotics and process control, where complex issues involving multivariate sensory information are present [5]. Furthermore, in robotics, self-organization allows autonomous construction of effective world representations either from raw sensory measurements or from preprocessed sensory data.

The contribution of this work to the field of unsupervised neural networks is threefold: (i) development of a time-delayed Hebbian learning rule to encode the temporal order of patterns in a sequence, (ii) use of temporal context to recall multiple stored sequences without ambiguity, and (iii) application of the proposed model to reduce the cost of robot "training" in tracking tasks. The learning algorithm to be proposed is evaluated through simulations of 2- and 3-dimensional robot trajectories.

This paper is organized as follows. In Section 2, we present some concepts related to the storage and retrieval of temporal sequences by means of neural network models. In Section 3, we develop our model discussing in details all its components. In Section 4, we evaluate the performance of the model through computer simulations and discuss the main results. Section 5 is devoted to compare the proposed model with others available in the literature on temporal sequences. We conclude the paper in Section 6, presenting possible directions for further developments.

2. Short-Term Memory in Temporal Sequence Based Control

Two ingredients are essential for autonomous reproduction of sequential patterns [19]. First, for the purpose of learning, a mechanism to extract and store transitions from one pattern to its successor in the sequence. This mechanism is known as *short-term memory* (STM). Second, for the purpose of recall, activation dynamics must be defined to mimic the previously learned sequence by propagation of the correct sequence of stored states.

In the context of temporal sequence processing, STM is the generic name of a number of retention mechanisms. STM aids temporal order learning and recall within a sequence by maintaining vestiges of such patterns for a certain period of time. Hence, an STM model can establish temporal associations between consecutive patterns and reproduce their order of occurrence at the network output.

There is a number of STM models within the framework of artificial neural networks [20–22]. The simplest one, called *tapped delay lines*, involves a buffer containing the most recent symbols from a sequence. Such a buffer consists of time delays serially connected. These lines convert a temporal sequence into a spatial pattern by concatenating the sequence components through a fixed-size window which slides in time. The concatenated vector is then presented to the network. Tapped delay lines are common in neural

network models and form the basis of traditional statistical autoregressive models [20]. For further details on the role of time delays in temporal sequence learning, the readers are referred to Herz [23].

The number of time delays defines the memory depth, i.e., the period of time a pattern remains available in the STM. For instance, four time delays indicate that a particular pattern and its four predecessors are available in the memory. The model to be proposed in the next section uses time delays at the input and the output. When connected to the input, time delays are used to account for past elements of the sequence in order to resolve potential ambiguities during recall. When linked to the output units, time delays are used to learn the temporal order of the items of the input sequence.

3. The Neural Model Description

In this section, we introduce an artificial neural network model according to the framework proposed by Rumelhart and McClelland [24]. In the subsections, we describe the input, the network topology, the network rules and procedures.

3.1. *About the Input Patterns*

The input patterns are in the form of sequences. Each sequence consists of a finite number of items, also called sequence states or components, which can be scalars, $x(t) \in \Re$, or vectors, $\mathbf{x}(t) \in \Re^p$, $p > 1$. These items, presented to the network sequentially, one after the other, represent the spatial portion of the input state, and the order in which they occur represent the temporal order. The network should be able to encode both the spatial and temporal aspects of the input sequence.

We classify sequences as *open* and *closed*. Open sequences are those in which the initial item is different from the final one. For closed sequences, the initial item is equal to the final one. For instance, the sequence of letters A-B-C-D-E is open whereas the sequence of letters X-Y-Z-W-X is closed.

A single open or closed sequence can have intermediate *repeated* items. For example, the sequences A-B-C-D-C-E and X-Y-Z-W-Z-X are examples of open and closed sequences with repeated items (C in the first sequence and Z in the second one), respectively. In addition, two or more sequences can have items in common. For instance, the sequences A-B-C-D-E and X-Y-C-W-Z share the item C. We call this item a *shared* state. Generically, we call both repeated and shared states *recurrent* items. Recurrent items cause ambiguities during recall, and, because of this, we use the term *complex* sequences for those with recurrent states. Some kind of contextual information should be supplied in order to resolve such ambiguities.

If more than one sequence is to be presented, in order to distinguish between the end of one sequence and the beginning of another, two alternatives are possible. The first one is to define a time delay between consecutive sequences. The second is to use a sequence identifier. Whenever this identifier changes, this means that the sequence has also changed. We have chosen the second alternative because it can be used as a form of context information that enables the network to handle ambiguities that occur when repeated items are present in the sequences. This type of context and another are described below.

For the robot trajectories, each state is composed of the spatial position (x, y, z) of the robot end-effector in its workspace, six joint angles and six joint applied torques.

3.2. *The Architecture*

The basic architecture of the proposed model is illustrated in Fig. 1. This is a two-layer network composed of a broadcasting input layer and an output layer responsible for the processing. The model has feedforward and feedback weights playing distinct roles in its dynamics. From this point onwards, the term trajectory is synonymous with sequence.

The input pattern entails two sets of neurons: the *sensory* and the *context* units. The sensory set, $\mathbf{s}(t) \in \Re^p$, receives the input trajectory state at time step t and propagates this vector towards the output

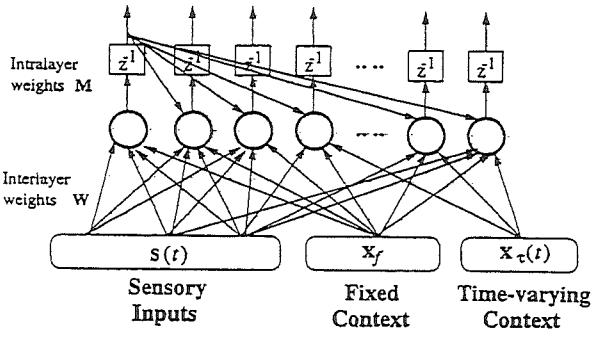

Figure 1. The architecture of the proposed model.

units. No input data preprocessing stage is required. The context units are used to resolve ambiguities that may occur during recall of complex trajectories. The context units are of two types: *fixed* and *time-varying*. Fixed context, $\mathbf{x}_f \in \Re^q$, is time-invariant and is set to a particular state of the temporal sequence, the initial or the final one being the usual options. It is kept unchanged until the end of the current sequence has been reached. Thus, it acts as a kind of a *global sequence identifier*. Time-varying context units change their state of activity every time a new input pattern is considered, and it is formed by the concatenation of *past sequence items*, $s(t-l) \in \Re^p$, $l = 1, \ldots, \tau$, where τ is called *memory depth* [20]. Thus, $\mathbf{x}_\tau(t) = \{s(t-1), \ldots, s(t-\tau)\}$, so that $\mathbf{x}_\tau(t) \in \Re^{\tau \cdot p}$. The sensory input, the fixed and the time-varying context are combined to form the input pattern, $\mathbf{v}(t)$, to be presented to the network at time t, i.e., $\mathbf{v}(t) = [s(t)\ \ \mathbf{x}_f\ \ \mathbf{x}_\tau(t)]^T$. Note that $\dim \mathbf{v}(t) = N_i = p + q + \tau \cdot p$, where N_i is the number of input units.

The current model extends previous work on context and temporal sequence learning for robot control [25, 26]. The previous architectures could deal only with open temporal sequences with shared items, because they made use only of fixed-type context. This type of context is unable to deal with closed trajectories with repeated states, such as figure-eight sequences. The solution is to include time-varying context units which take into account the past history of the sequence, allowing the network to encode both closed and open trajectories with recurrent items.

The synaptic weights consist of *feedforward* (or *interlayer*) weights and *feedback* (or *intralayer*) weights. The feedforward weights connect the input units to the output neurons. These connections store the items of a particular sequence through a competitive learning rule. That is, for a particular sequence item, a single output neuron (the winner) or a small group of output neurons are chosen to store this sequence item. The feedback set of weights indicates the temporal order of the patterns in a sequence by using a Hebbian learning rules to form temporal associations from the previous to the current winner of the competition. Feedback weights are unidirectional and initialized with zeros for the training phase, indicating no temporal association at all. Also, there is no feedback self-connection, i.e., a connection from the output of a neuron to its input.

The output neurons represent the current and the next states in a particular sequence. The current state is stored in the weight vector of the neuron with the highest value for $a_j(t)$, $j = 1, \ldots, N_o$, where N_o is the number of output neurons. The next state is stored in the weight vector of the neuron with the highest value for $y_j(t)$, $j = 1, \ldots, N_o$ (Eq. (4) in Section 3.3). This weight vector is then used as a control signal, to position the robot arm at the desired configuration.

3.3. *Activation and Output Rules*

The two groups of synaptic weights presented in the last section are updated during a *single* pass of an entire trajectory in which each sequence item is presented only once. This means that a sequence with N_c components requires *exactly* N_c training steps. Thus, following the presentation of a sequence item, this input pattern is compared with each feedforward weight vector, using a measure of dissimilarity based on Euclidean distance, and the group of weight vectors closest to the input vector is selected to be updated. Mathematically, we have:

$$\begin{aligned}
\nu_1 &= \arg\min_j \{f_j(t) \cdot \|\mathbf{v}(t) - \mathbf{w}_j(t)\|\} \quad \forall j \\
\nu_2 &= \arg\min_j \{f_j(t) \cdot \|\mathbf{v}(t) - \mathbf{w}_j(t)\|\} \quad \forall j \notin \{\nu_1\} \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\nu_N &= \arg\min_j \{f_j(t) \cdot \|\mathbf{v}(t) - \mathbf{w}_j(t)\|\} \\
&\qquad \forall j \notin \{\nu_1, \ldots, \nu_{N-1}\}
\end{aligned} \tag{1}$$

where $\{\nu_1, \ldots, \nu_N\}$ are the indices of the output neurons ranked according to the proximity between their weight vectors and the current input; thus, ν_1 is the index representing the neuron whose weight vector is the closest option to the current input vector. When the parameter K, called *degree of redundancy*, exceeds one, we have a population of neurons encoding a single vector of an input sequence; in other words, a redundancy mechanism. On one hand, such a scheme, similarly to neighboring neurons in the Kohonen SOM, allows the network to be robust, i.e., to be tolerant to noise and neuron failure. On the other hand, the redundancy mechanism increases memory requirements. For the purpose of learning, we usually set $K > 1$. For recall, we always set $K = 1$.

The function $f_j(t)$, called *the exclusion factor*, is defined as:

$$f_j(t+1) = \begin{cases} \alpha & \text{if } j \in \{\nu_1, \cdots, \nu_K\} \\ f_j(t) & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha \gg 1$ and $f_j(0) = 1$, $j = 1, \ldots, N_o$. This function is used to "exclude" the K winning neurons from subsequent competitions, to ensure that each point of the trajectory is encoded by different neurons. The exclusion mechanism is akin to that proposed by James and Mikkulainen [27]. However, their model aimed at detecting a single sequence instead of recalling sequential patterns. Furthermore, they did not propose a mathematical formalism for their exclusion mechanism.

The combination of redundancy and exclusion mechanisms yields a unique group of neurons to represent a specific state of the trajectory. Such groups are linked in the correct temporal order through a lateral coupling structure. The neuron activations are determined by the following equation:

$$a_{v_i}(t) = \begin{cases} A_{\max} \cdot \gamma^{i-1} & \text{for } i = 1, \ldots, K \\ 0 & \text{for } i > K \end{cases} \quad (3)$$

where $0 < \gamma < 1$ is an activation decay term, and $A_{\max} \geq 1$ is the maximum activation value obtained for $i = 1$. According to Eqs. (1) and (3), the closer the weight vector to the current input vector, the higher the activation of the associated neuron. Once a neuron is active, its activation is diffused through a non-zero lateral connection in order to trigger its successor in the current sequence. The largest output value $y_j(t)$ determines the weight vector to be sent to the robot controller:

$$y_j(t) = g\left(\sum_{r=1}^{N_o} m_{jr}(t) a_r(t)\right) \quad \text{for } j = 1, \ldots, N_o \quad (4)$$

where $g(\cdot)$ is a function defined so that $g(u) \geq 0$ and $dg(u)/dt > 0$, and $m_{jr}(t)$ is the intralayer connection weight between the output neurons r and j.

3.4. *The Learning Rules*

After the selection of the winning neurons and the determination of their activations and outputs, the weight vectors $\mathbf{w}_j(t)$ are updated according to the following competitive learning rule [28]:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \delta(t) a_j(t)[\mathbf{v}(t) - \mathbf{w}_j(t)] \quad (5)$$

where $\delta(t) \approx 1$ is the learning rate. This competitive learning procedure copies the input vector $\mathbf{v}(t)$ to the weight vectors of the K winning neurons obtained through Eq. (1). Note that units with activations $a_j(t)$ equal to zero do not learn at time step t.

Without the exclusion mechanism, the competitive rule in Eq. (5) would try to group the sequence items in clusters, reducing the number of states of the input sequence. Since our goal is to reproduce exactly the same sequence at the network output, this clustering effect should be avoided.

It is worth remembering that the input vector $\mathbf{v}(t)$ is comprised of three parts: a sensory part corresponding to the sequence item currently being observed, the fixed context and the time-varying context. We have the following two situations: (1) A single open or closed sequence contains a repeated item: the first time this item occurs, a particular neuron will store the corresponding input vector in its synaptic weights. When the item occurs for the second time, the sensory part and the fixed context are equal to that of the first occurrence of the repeated item, since the sequence is the same, but the time-varying context is different since it consists of the τ immediate predecessors of the current sequence item. (2) Multiple open sequences share an item: using arguments similar to case (1), every time the shared item reoccurs, the sensory part remains the same but the fixed and time-varying context are different. This way, the network is able to recall the stored sequences without ambiguity, since the repeated and shared states are stored in the feedforward weights together with their corresponding contexts.

The intralayer weights are updated according to the following rule:

$$\Delta m_{jr}(t) = \lambda a_j(t) a_r(t-1) \quad (6)$$

where $0 < \lambda \leq 1$ is the intralayer learning rate. According to Eq. (6) lateral connections will be established from the winners of the previous competition, $r = \{v_1(t-1), v_2(t-1), \ldots, v_K(t-1)\}$, to the winners of the current competition, $j = \{v_1(t), v_2(t), \ldots, v_K(t)\}$. Figure 2 sketches how Eq. (6) learns the temporal order for the simplest case, in which $K = 1$. Initially ($t = 0$), the network has no lateral connections. At $t = 1$, the neuron on the left is the winner for the pattern $\mathbf{v}(1)$. At $t = 2$, the neuron on the right is the winner for pattern $\mathbf{v}(2)$. Still at $t = 2$, a lateral connection is created from the neuron on the left to the neuron on the right through Eq. (6), learning the transition $\mathbf{v}(1) \rightarrow \mathbf{v}(2)$. This process continues until all transitions between successive sequence items is learned.

Some brief comments are necessary at this point. First, the neuron activations of the previous

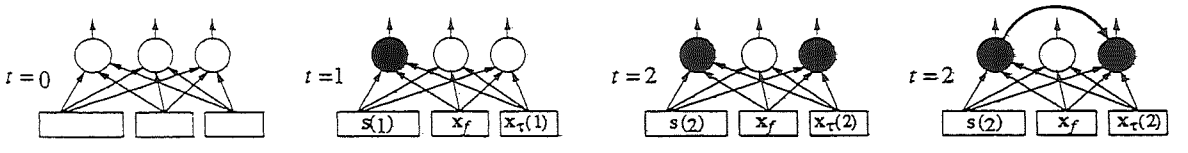

Figure 2. A sketch of how consecutive winners are temporally linked through lateral connections. $\mathbf{x}_f$ is the fixed context and $\mathbf{x}_\tau(t)$ denotes the time-varying context.

competition, $a_r(t-1)$, are made available through time delays (STM model). Second, Eq. (6) is an asymmetric Hebbian learning rule [29] which aims at creating temporal associations between consecutive patterns in the input trajectory. Indeed, this equation encodes the *temporal order* of the input sequence.

3.5. *Temporal Order Learning and One-Step-Ahead Recall*

The simple form of Eq. (6) allows the construction of a hypothetical example based on the concept of *temporal associative memory* [30] to elucidate temporal order learning and one-step-ahead recall. Thus, Eq. (6) can be written in matrix form as follows:

$$\mathbf{M}(t+1) = \mathbf{M}(t) + \lambda \mathbf{a}(t)\mathbf{a}^T(t-1) \qquad (7)$$

where $\mathbf{M}(t+1)$ is the *feedback memory matrix* corresponding to the learning of one state transition given by the activation pair $(\mathbf{a}(t), \mathbf{a}(t-1))$. For a sequence with N_c items, the resulting matrix is:

$$\mathbf{M}(N_c) = \mathbf{M}(0) + \lambda \sum_{\tau=1}^{N_c} \mathbf{a}(\tau)\mathbf{a}^T(\tau-1) \qquad (8)$$

Note that this matrix is constructed in an incremental manner, i.e., it cannot be set in advance as in other associative memory models, since the activation patterns $\mathbf{a}(t)$ are not known beforehand.

As already mentioned, the activation patterns $\mathbf{a}(t)$ indicate the neuron whose weight vector best matches with the current input item, and the output patterns $\mathbf{y}(t)$ indicate the neuron whose weight vector stores the next sequence item. The recall of the next item depends on the feedback memory matrix and on the current activation pattern (see Eq. (4)). The following hypothetical example illustrates this property.

Consider a trajectory with only three states ($N_c = 3$) and a network with three neurons ($N_o = 3$). Setting $K = 1$, we assume that neuron $j = 1$ encoded the first state of the trajectory at $t = 1$, neuron 3 encoded the second state at $t = 2$, and neuron 2 encoded the third state at $t = 3$. Hence, the corresponding activation patterns were $\mathbf{a}(1) = [1\ 0\ 0]^T$, $\mathbf{a}(2) = [0\ 0\ 1]^T$ and $\mathbf{a}(3) = [0\ 1\ 0]^T$. Thus, in accordance with Eq. (8), the learned feedback memory matrix is:

$$\begin{aligned}
&\mathbf{M}(N_c) \\
&= \mathbf{M}(0) + \lambda\{\mathbf{a}(3)\mathbf{a}^T(2) + \mathbf{a}(2)\mathbf{a}^T(1) + \mathbf{a}(1)\mathbf{a}^T(0)\} \\
&= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \lambda \left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} (0 \quad 0 \quad 1) \right. \\
&\quad \left. + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} (1 \quad 0 \quad 0) + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} (0 \quad 0 \quad 0) \right\} \\
&= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \qquad (9)
\end{aligned}$$

To illustrate how the feedback memory matrix constructed by Eq. (9) retrieves the next sequence item, consider the following function: $g(u) = u$, for $u \geq 0$ and $g(u) = 0$, otherwise. Note that, in Eq. (4), $\sum m_{jr} a_r > 0$, then we have the following linear relationship for recall purpose: $\mathbf{y}(t) = \mathbf{M}\mathbf{a}(t)$. Thus, if the first sequence item is presented again, the resulting activation pattern is $\mathbf{a}(1) = [1\ 0\ 0]^T$.

Sequence recall is initiated by giving a pattern in the sequence as a *cue stimulus*; then, the part of the sequence that follows the cue pattern is successively recalled. The output pattern is obtained as follows:

$$\mathbf{y}(1) = \mathbf{M} \cdot \mathbf{a}(1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \lambda \end{pmatrix}$$

which indicates that neuron $j = 3$ stored the next trajectory state in its weight vector. This weight vector supplies the robot controller with the next spatial

position, the associated joint angles, and the joint applied torques. Once a robot has reached its next position, new sensor readings are fed back to the neural network input to produce the following activation pattern $\mathbf{a}(2) = [0\ 0\ 1]^T$. The corresponding next sequence item is then:

$$\mathbf{y}(2) = \mathbf{M} \cdot \mathbf{a}(2) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \lambda \\ 0 \end{pmatrix}$$

which indicates that neuron $j = 2$ stored the last trajectory state in its weight vector. When the robot arm reaches its final position, the new sensor readings together with context information produce the activation pattern $\mathbf{a}(3) = [0\ 1\ 0]^T$. The next sequence item corresponding is then:

$$\mathbf{y}(3) = \mathbf{M} \cdot \mathbf{a}(3) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

which indicates that the trajectory has indeed reached its end, because there is no "next item".

4. Simulations

This section aims at evaluating the proposed neural network in terms of storage and recall of different types of trajectories, as well as how the network parameters affect the overall performance of the system. First, we consider closed 2D trajectories (circular and figure-eight types), and then, multiple 3D robot trajectory processing is assessed.

The open and closed trajectories were generated by the *ROBOTICS* toolbox of Matlab [31], for a PUMA 560 robot with 6 DOF. These trajectories were previously used to evaluate recurrent [10] and associative memory neural models [32] in temporal-sequence-based control of robotic arms. As pointed out by Wang and Yuwono [33], learning of multiple sequences can be carried out with simultaneous or sequential input presentations, and the latter was chosen in our case. By convention, the robot movements are executed within a cube of dimension $1m \times 1m \times 1m$. The origin of a coordinate frame for the robot end-effector is located at the center of the cube.

Closed trajectories, included in this study, are commonly used as benchmarks for sequence processing [34–36]. For the circular trajectories, we have sequences with 20, 35, 70 and 100 states. For the figure-eight trajectories the sequences are 20, 40, 80 and 100 states long. The open trajectories were used to test the ability of the network to work with multiple trajectories with shared states. Each open trajectory has 11 states, including the initial and the final ones.

In both open and closed trajectories, each state is constituted by the spatial position (x, y, z) of the robot end-effector in its workspace, six joint angles and six joint applied torques. Thus, $p = 3 + 6 + 6 = 15$. The fixed context is set to the target position of the end-effector (final state of the trajectory), and thus $q = 3$. The time-varying context consists of past end-effector positions, and it has depth $\tau = 1$. Then, the total number of input units is $N_i = p + q + \tau \cdot p = 15 + 3 + 15 = 33$.

The network performance is evaluated in tracking tasks by means of the root mean square error (RMSE) given by the following equation:

$$RMSE(N_c) = \sqrt{\frac{1}{N_c} \sum_{t=1}^{N_c} \left(x_d^t - x_r^t\right)^2 + \left(y_d^t - y_r^t\right)^2 + \left(z_d^t - z_r^t\right)^2}$$

where N_c is the number of patterns in a trajectory, (x_d, y_d, z_d) and (x_r, y_r, z_r) are the desired and recalled coordinates of the robot end-effector. These coordinates are obtained from the first three components of the input and winner weight vectors at time step t.

4.1. *Learning of Closed Trajectories*

In the following paragraphs we evaluate the influence of network parameters on the network performance during the learning of closed circular and figure-eight trajectories. The following tests include: choice of learning rate δ, influence of redundancy on fault-tolerance, and influence of sampling rate and redundancy on noise-tolerance.

4.1.1. Choice of Learning Rate δ. In this simulation, intended to show how the learning rate influences the storage accuracy of the proposed model, we trained the network on the circular and figure-eight trajectories with four different values of the feedforward learning rate δ: 0.45, 0.75, 0.90 and 0.99. The other parameters were set to the following values: $K = 1$, $\alpha = 10^6$, $\lambda = 0.8$, $A_{max} = 1$, $\gamma = 0.99$, and $N_o = 100$. The feedforward weights were randomly initialized between 0 and 1, the feedback units were initialized to zero, and the same initial weights were used for all values of δ.

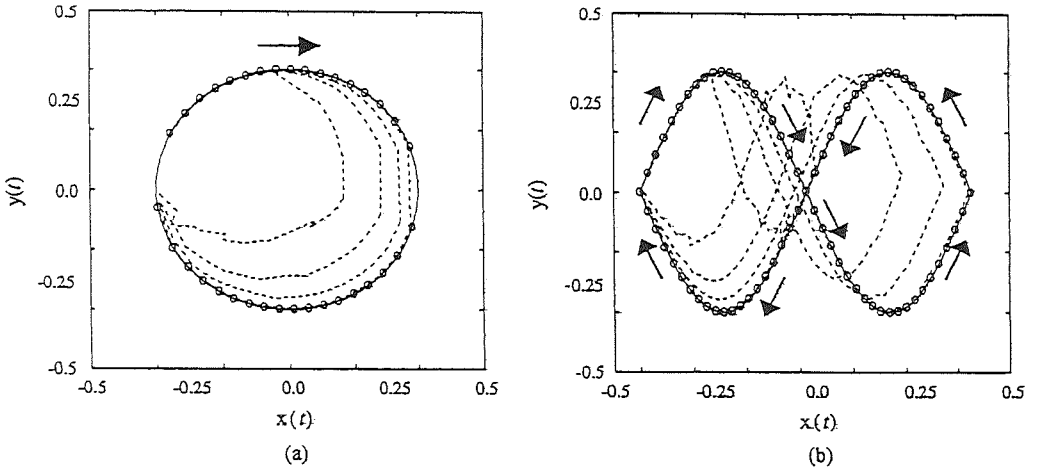

Figure 3. Accuracy in learning closed trajectories for $\delta = 0.45$, 0.75, 0.90 and 0.99. Inner trajectories have lower values for the learning rate δ. Arrows indicate the direction of movement.

The resulting trajectories are plotted in Fig. 3(a) for a circular trajectory with 35 discrete patterns. Figure 3(b) gives the general behavior for a figure-eight trajectory with 80 points.

The errors for the circular trajectories were: 2.173124 for $\delta = 0.45$, 0.965519 for $\delta = 0.75$, 0.377616 for $\delta = 0.90$, and 0.038847 for $\delta = 0.99$. The errors for the figure-eight trajectory were: 12.030423 for $\delta = 0.45$, 5.409065 for $\delta = 0.75$, 2.171650 for $\delta = 0.90$, and 0.218585 for $\delta = 0.99$. These figures indicate that the RMSE decreases as δ increases. Hence, to achieve accuracy, δ must be near or equal to 1. This is an important requirement since the robot controller must be supplied with precise signals from the network.

4.1.2. Influence of Redundancy on Fault-Tolerance. In this simulation, we show how a trajectory is stored by the first K winning neurons, and why such a redundancy mechanism is useful in cases of neuron failure. We chose $K = 3$, which means that each point of the sequence is encoded by 3 different neurons. The other parameters were set to the following values: $\delta = 1, \alpha = 10^6, \lambda = 0.8, A_{max} = 1, \gamma = 0.99, \tau = 1$, $N_o = 525$. The results for a circular trajectory with 70 points are shown in Fig. 4. Figure 4(a) illustrates the input (circles) and the stored/retrieved trajectory (crosses) encoded by the first winner neuron, while Fig. 4(b) presents the result for the third winner unit.

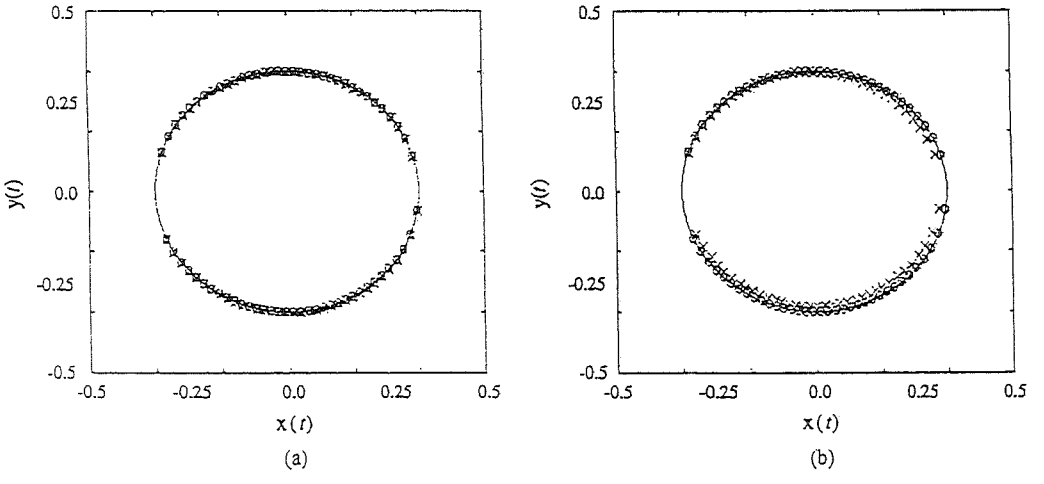

Figure 4. Effects of redundancy on the learning of circular trajectories by the: (a) 1st winner (higher activation) and (b) 3rd winner (lower activation).

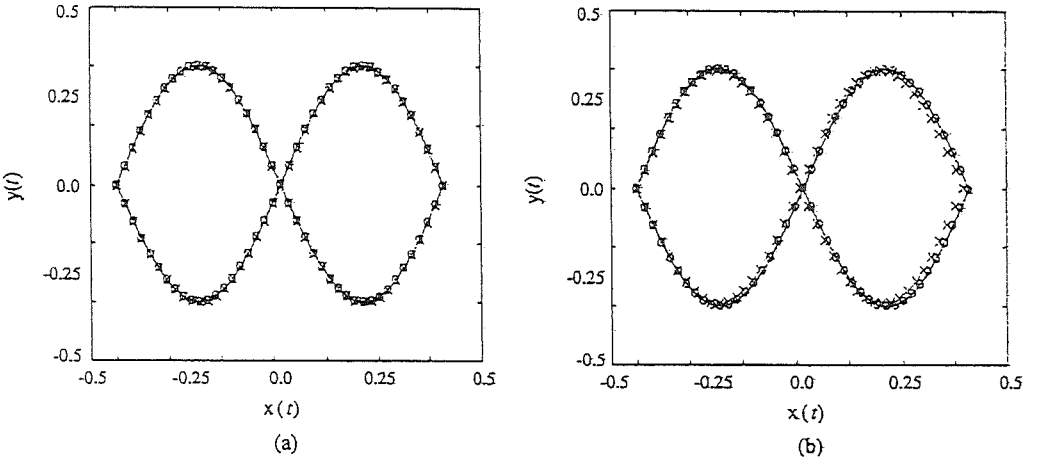

Figure 5. Effects of redundancy on the learning of figure-eight trajectories by the: (a) 1st winner (higher activation) and (b) 3rd winner (lower activation).

The resulting RMSE values for the retrieved trajectories were 0.00 (1st winner), 0.143867 (2nd winner), and 0.287732 (3rd winner). The RMSE values for the second and third winners can be viewed as worst cases. For example, if all the first winners have collapsed, the second winners would be used instead, yielding RMSE = 0.143867. In the extreme and unlike case of total collapse of the first and second winners, the third would be used by the network, yielding RMSE = 0.287732. Isolated neuron failures would result in intermediate values for RMSE.

An example of a figure-eight trajectory with 80 points is plotted in Fig. 5. This sequence has a crossing position at coordinates (0.0, 0.0), which explains the need for temporal context information. To recall the trajectory in the correct way, the time-varying context units are set to the coordinate of the pattern which immediately precedes the current sensory input. The resulting RMSE values were: 0.00 (1st winner), 0.223320 (2nd winner), and 0.445203 (3rd winner). We can conclude that, for the purpose of tracking, the robot controller must use the trajectory in Figs. 4(a) and 5(a). In the case of neuron failure, the stored trajectories will continue to be retrieved at the expense of a slightly higher RMSE value.

It is worth noting that the network can store and retrieve a trajectory with RMSE = 0 even in the presence of neuron failures, by simply adopting $\delta = \gamma = A = 1$. However, this would make the network much like a fault-tolerant conventional storage-and-recall device (look-up table) without the ability to respond well to noisy sequences, which is a highly desirable network property.

4.1.3. Influence of Sampling Rate and Redundancy on Noise-Tolerance. The simulations considered in the previous sections handled noise-free trajectories. However, tolerance to noise is a desirable property for any controller of a real robotic system. This network property was evaluated by adding different amounts of zero mean Gaussian white noise to the trajectory patterns and calculating the RMSE value. The noise had variance levels ranging from 0.001 to 0.1.

A related issue is the effect of the sampling rate (number of points in a sequence) on the network performance [36]. Hence, in this test, we aimed to evaluate how the network responds to a noisy trajectory while varying the degree of redundancy and the number of items of the input trajectory.

We simulated the network for three values of degree of redundancy: $K = 3$, 4 and 5. Figures 6 and 7 show the results for circular and figure-eight trajectories, respectively. It can be seen in Fig. 6 that lower values for RMSE (solid lines) are obtained by choosing $K = 5$. The worst results were obtained for $K = 3$ (dashed-dotted lines) and 4 (dotted lines). However, as the value of K increases, the improvement in RMSE is less patent.

In addition, these results clearly indicate that the RMSE rises as the number of points in a sequence is increased. This can be explained by noting that as the distance between consecutive points decreases at higher sampling rates, the chance of the network choosing

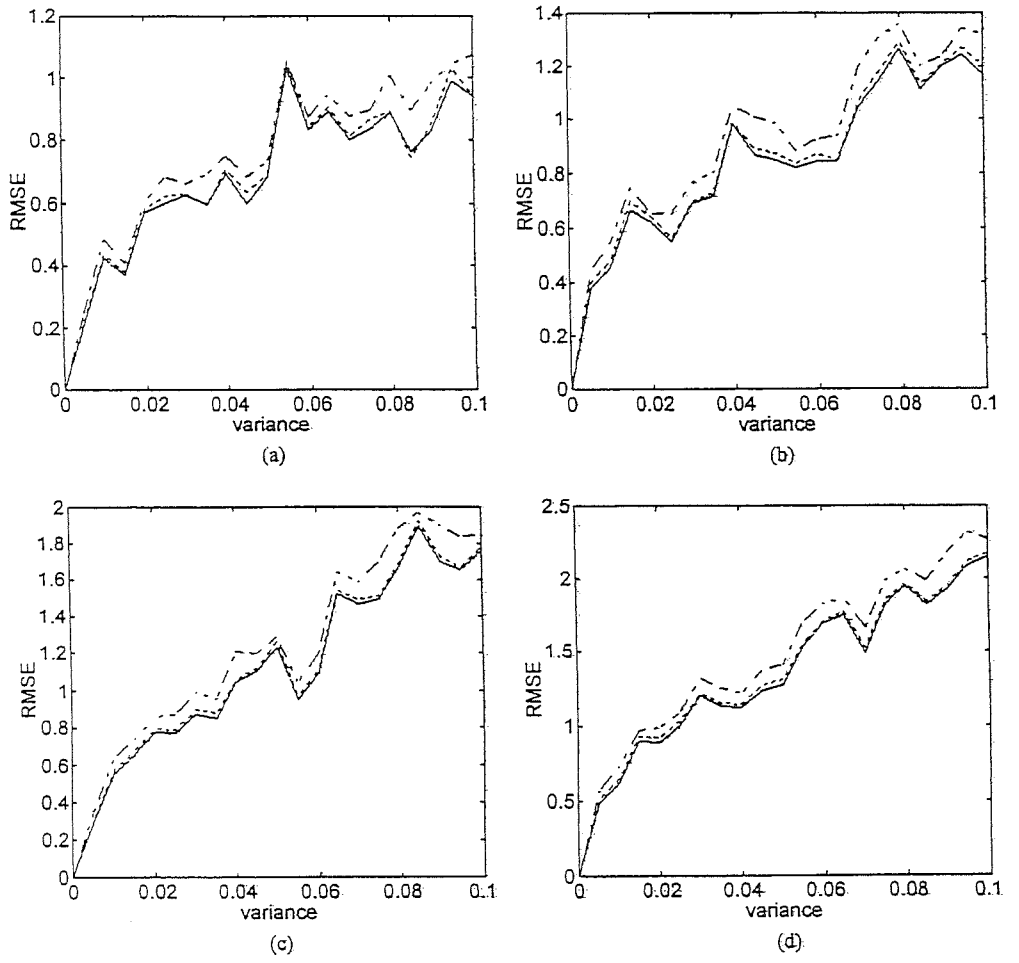

Figure 6. Noise-tolerance of the network trained on circular trajectories for different sampling rates: (a) 20 points, (b) 35 points, (c) 70 points and (d) 100 points.

an incorrect winner due to noise increases. This result contrasts with previous simulations encountered in the literature [35] in which increasing sampling rates results in higher resilience to noise.

So far, the results obtained suggest that the network gains in robustness by using a redundancy degree $K > 1$ (we suggest $K = 2$ or $K = 3$). Also, it is useful to have $\gamma < 1$, which affords some noise tolerance. Another important property of the proposed model, the ability to store and recall with multiple trajectories, is studied in the next section.

4.2. *Learning of Multiple Robot Trajectories*

In order to test the ability of the algorithm to encode multiple trajectories, the following assumptions were made: (1) the initial and final points of a given trajectory are known and (2) any trajectory must contain at least one crossing point with all the others. In the current work, we focus on trajectories with one common point which can be situated at any intermediate position. The network parameters were set to $\alpha = 1000$, $A_{max} = 1$, $\gamma = 0.95$, $\delta = 1.0$, $\lambda = 0.8$, $\tau = 1$, $N_o = 70$, and three trajectories were trained sequentially.

Trajectories with at least one point in common suffer the *perceptual aliasing* problem. In the present work, this problem is stated as: "which trajectory should the arm follow subsequent to a point belonging to more than one?" This problem is solved by the proposed model through the use of context (Section 3.2). Figure 8 shows the network results following the training stage on three trajectories. Trajectories in Fig. 8(a) and (b) have a crossing point at (0.20, 0.30, 0.0) and those in

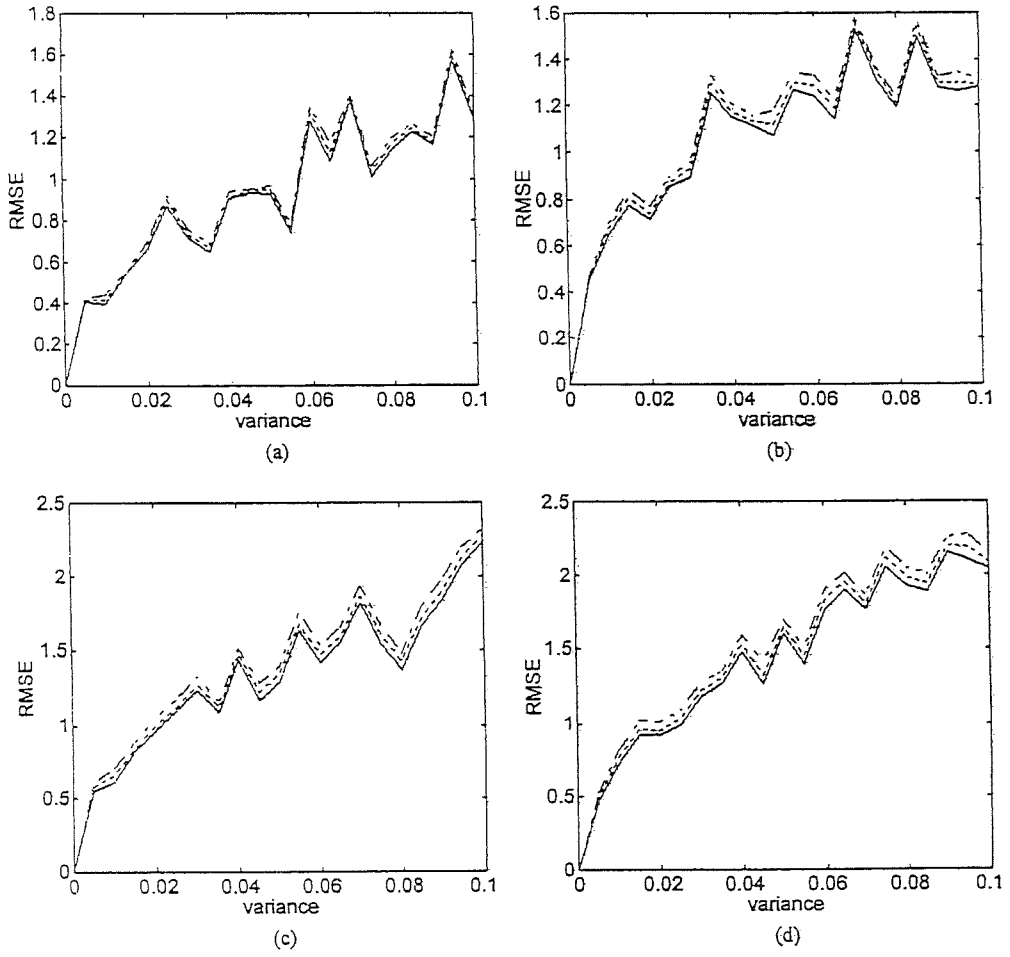

Figure 7. Noise-tolerance of the network trained on figure-eight trajectories for different sampling rates: (a) 20 points, (b) 40 points, (c) 80 points and (d) 100 points.

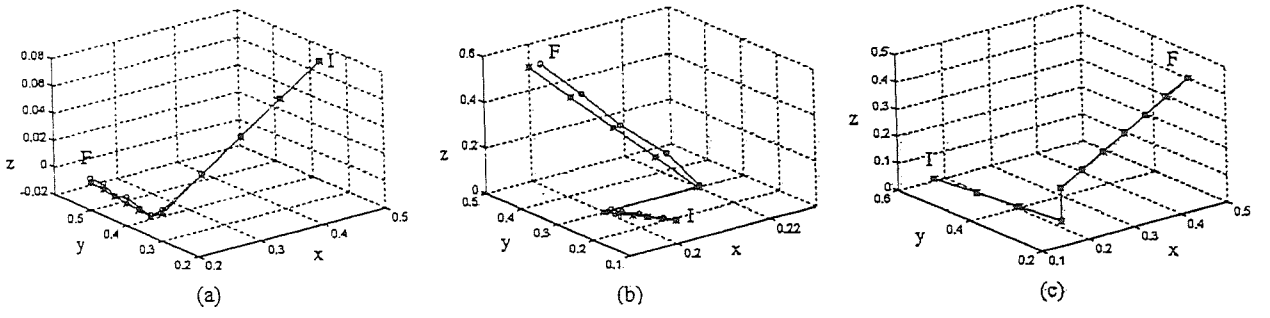

Figure 8. Three learned trajectories with one point in common. A desired trajectory is represented by open circles and a retrieved trajectory is represented by asterisks.

Fig. 8(b) and (c) have a crossing point at (0.22, 0.30, 0.0). It is worth noting that the stored and the desired trajectories in all cases are very similar. For example, the RMSE value obtained for the trajectory in Fig. 8(a) is 0.0024. This illustrates the ability of Eq. (5) to encode an input pattern accurately in only one iteration. The letters I and F indicate the initial and final points of the trajectory, respectively.

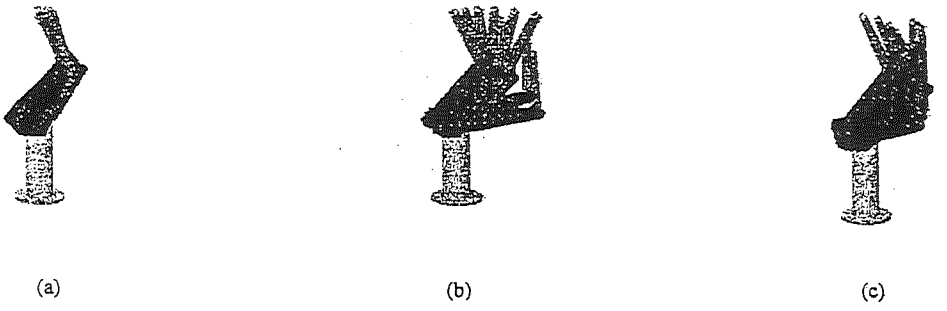

Figure 9. The retrieved trajectories in Fig. 8 in a simulated robot workspace.

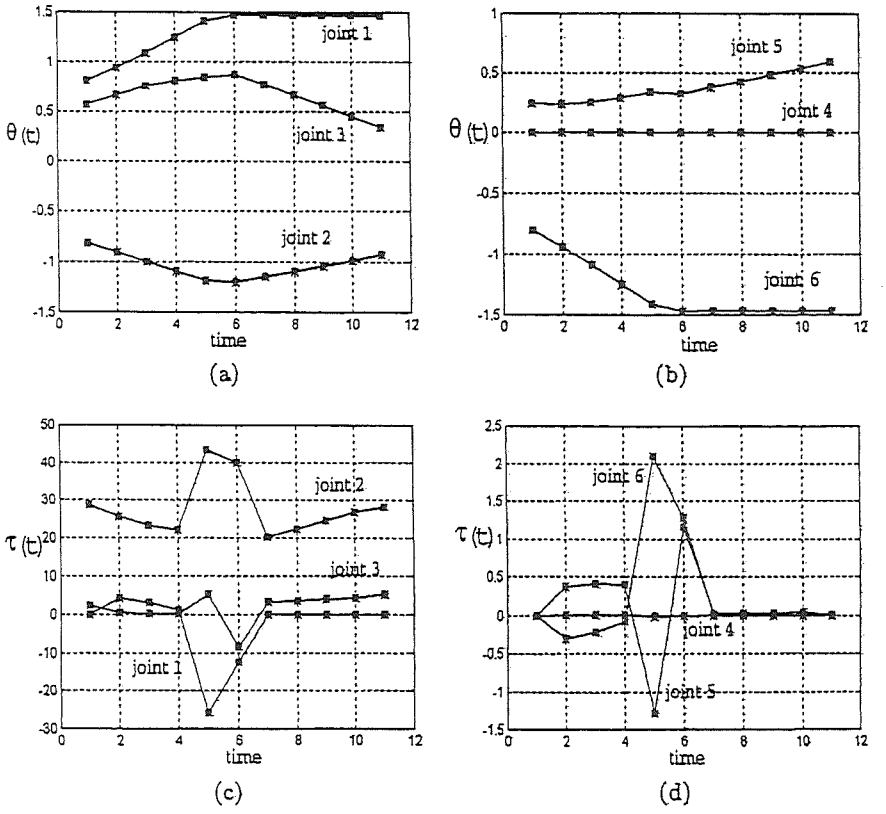

Figure 10. The joint angles (a) and (b), and torques (c) and (d) associated with the points of the trajectory shown in Fig. 8(a). Angles in radians and torques in N.m.

Figure 9 shows the recalled trajectories in Fig. 8 in a simulated robot workspace, based on the Simderella simulator [37]. This simulated environment follows the correct relative dimensions of a typical PUMA 560 robot.

Figures 10–12 show the joint angles and torques associated with each point in the stored trajectories. Similarly, the algorithm was able to encode them with a small error, since the desired and stored values are practically the same. Note that the algorithm can learn the input independently of its magnitude and sign, and it responds equally well to trajectories with smooth curvature (circular and figure-eight sequences) and with abrupt changes of direction (see Fig. 8).

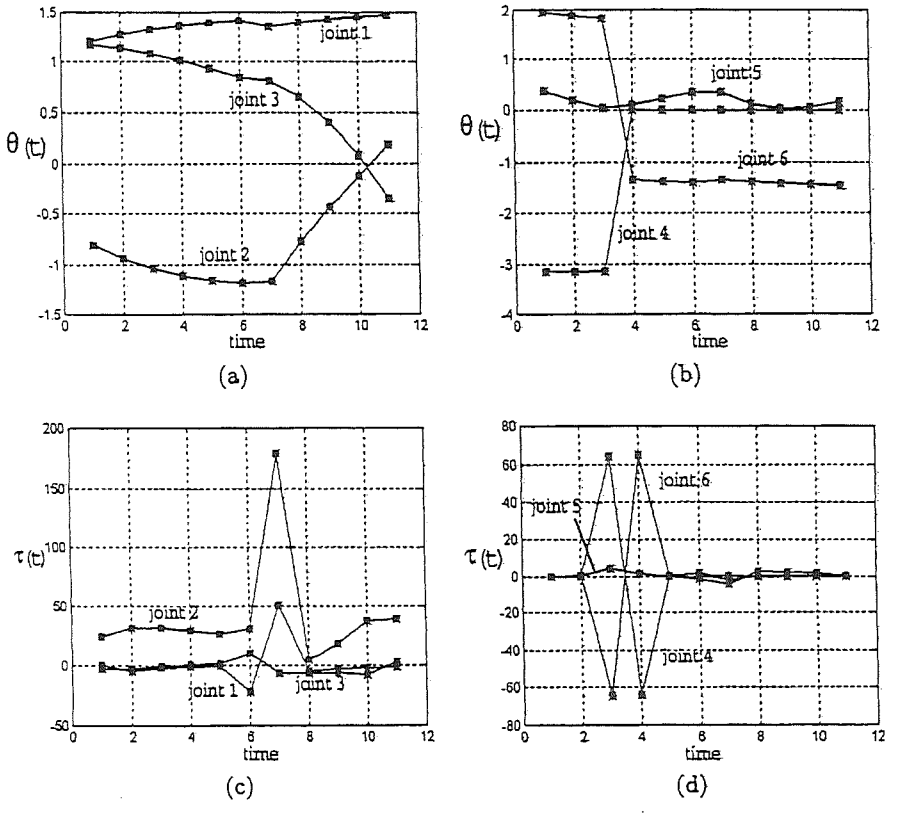

Figure 11. The joint angles (a) and (b), and torques (c) and (d) associated with the points of the trajectory shown in Fig. 8(b).

Figure 13 illustrates fault-tolerance for this type of trajectory. In this test, we simulated neuron faults in the same way as for circular and figure-eight trajectories, i.e., by excluding the first winning neurons, $\nu_1(t)$, for each item of the three trajectories. Even so, the network is able to reproduce the trajectories correctly at the expense of a slightly larger RMSE error, since the second winners, $\nu_2(t)$, are now responsible for the retrieval of the stored sequence. This result justifies the use of more than one neuron during the learning of the feedforward weights.

Despite the simplicity of the model, the simulations suggest that multiple trajectories can be learned very fast and accurately, independently of their complexity. Trajectories with more than one crossing point are analogously learned with small tracking error. In the next section we summarize the gains and limitations of the proposed model and discuss those aspects in which it differs from previous ones in the literature for temporal sequence learning and robot trajectory tracking.

5. Discussion

The proposed self-organizing neural network raises a series of important issues regarding the temporal sequence learning problem. In the following paragraphs, we present and discuss some of them in order to highlight the advances achieved by this model on existing neural network models, unsupervised or not, used to temporal sequence processing.

5.1. *The Chaining Hypothesis*

The basic idea of the chaining hypothesis is to view a temporal sequence as a set of associations between consecutive components, and learn these associations for later recall. This temporal association paradigm is widely used in many neural models. The vast majority of these models are based on either multilayer perceptrons (MLP) with some temporal version of backpropagation training [38] or the Hopfield model of

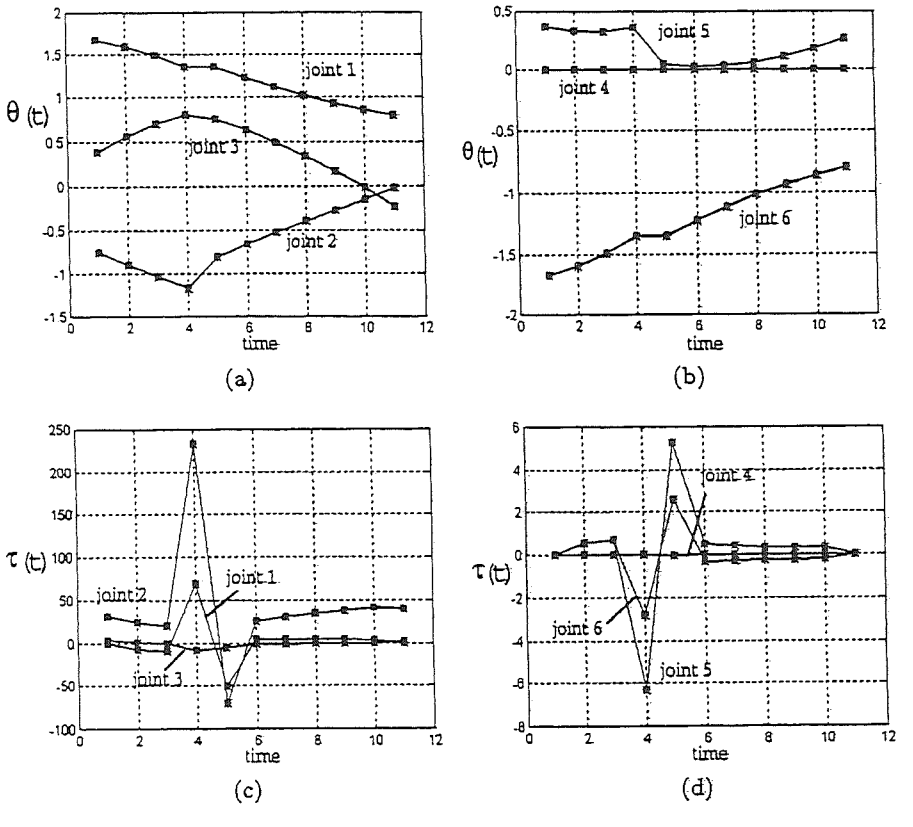

Figure 12. The joint angles (a) and (b), and torques (c) and (d) associated with the points of the trajectory shown in Fig. 8(c).

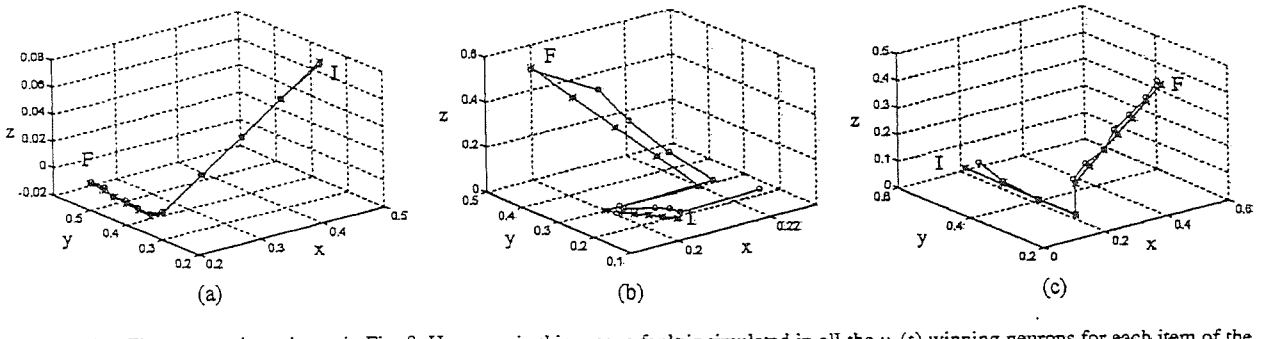

Figure 13. The same trajectories as in Fig. 8. However, in this case, a fault is simulated in all the $\nu_1(t)$ winning neurons for each item of the three trajectories.

associative memory [21, 33]. Also, BAM-type [32, 39] and ART-type [40, 41] model use the chaining hypothesis to recall temporal sequences. The model proposed in this paper also follows this paradigm; however, in contrast to those models based on MLP and BAM, it learns temporal associations in a self-organized manner and the learning process is considerably faster. Comparing the current model to other self-organizing ones such as those proposed by Grossberg [42] and Healy et al. [40], one can see that: (i) these models have difficulties in handling closed sequences with repeated points; (ii) they do not address the problem of fault tolerance and noise robustness. Furthermore, Healy et al. [40] coupled two ART 1 modules [43] trained on the same items of a sequence and associated the items learned from one ART 1 with those learned by the other

whereas our model uses a single layer of feedforward weights where the items are linked in the correct temporal order by lateral connections.

5.2. *The Temporal Context*

Context plays a crucial role in learning, especially when a subject or an artificial system has to handle ambiguous situations. The conventional approach to including temporal context into neural networks is to extend previous unsupervised models for static patterns by incorporating some type of STM. For this purpose, the most commonly used are tapped-delay lines and leaky-integrator neurons (see [22] for a review). We have adopted the "tapped-delay lines" approach for the time-varying context, but leaky-integrator neurons can be used alternatively. A drawback of the proposed model is that the depth τ of the time-varying context is non-adaptive, i.e., it has to be determined before learning takes place. The unsupervised model by Wang and Yuwono [33] learn the length of the temporal context necessary to recall sequences without ambiguity. It is important to note that temporal context as part of the network input also plays a fundamental role in classification of temporal patterns [44].

5.3. *Time-Delayed Hebbian Learning*

It is well known that time in Hebbian learning rules plays an essential role in psychology [45, 46], object recognition [47], route learning and navigation [48] and blind source separation [49]. To our knowledge, the proposed model together with that in Barreto and Araújo [25, 26] are the first to apply a time-delayed Hebbian learning rule to robotics. Similarly to our approach, Kopecz [50] used the concept of spatial items linked via a time-delayed Hebbian rule, to learn and recall temporal sequences. However, his model does not handle sequences with recurrent items.

5.4. *Application to Robotics*

Our model works much like a look-up table, since an input sequence item is associated with an output neuron, where extra information is available in its feedforward and feedback weights. Such information consists of control variables, such as the next spatial position of the end-effector, and the corresponding joint angles and torques, which are learned adaptively. Furthermore, the proposed model offers some degree of tolerance to noise and faults, an issue not easily addressed in conventional approaches to robot control via the look-up table method. Finally, this model handles ambiguous situations as easily as non-ambiguous ones, implying that it can be scaled-up to deal with more complex trajectories without additional difficulty.

As pointed out at the beginning of the paper, the walk-through method used in robot trajectory tracking tasks can become time-consuming and uneconomical. This occurs in part because the robot is out of production during the trajectory-learning process, and in part because, as the trajectories become more and more complex, the robot human operator may face difficulties in resolving ambiguities. The latter motivated strongly the development of the self-organizing neural network model presented in this paper, since it is highly desirable to have the trajectory-learning process automated with minimal human supervision.

When compared to other neural networks for trajectory tracking, the proposed model performs better than those of Hyötyniemi [51], Althöfer and Bugmann [11] and Bugmann et al. [6], because the temporal associations in those models are hard-wired. In the last two networks mentioned, two layers of connections store exactly the same components, and the temporal links are established by the network designer since the trajectory is known beforehand. For sequences with repeated or shared states, the first two models are unable to reproduce the stored trajectories correctly. The third can recall a single trajectory with repeated states but is unable to deal with multiple trajectories with shared states. Finally, as pointed out by Chen et al. [4], the issue of learning of multiple robot trajectories is usually neglected when considering neural networks models for tracking. This happens partly because of the inherent difficulties in dealing with recurrent states. Our model is the first unsupervised one proposed to handle tracking of multiple open and closed trajectories with repeated and shared states.

6. Conclusion and Further Work

An unsupervised model for learning and recall of temporal sequences is developed in this paper and applied to robot trajectory tracking. The simple neural network model accurately stores and retrieves complex sequences. An important advance introduced in this article is the processing of multiple sequences by our model.

In addition, the proposed temporal-sequence-based control system has other properties that are of great importance to the design of intelligent robotic systems: (i) noise tolerance, (ii) ability to learn multiple trajectories incrementally, (iii) ability to handle trajectories sampled at different rates, (iv) fault tolerance, (v) ability to learn open and closed trajectories with repeated and shared items, and (vi) ability to recall a learned trajectory from any intermediate point.

A foreseen limitation of our model is related to the storage of long sequences. Since the redundancy and exclusion mechanisms demand relatively high number of output neurons, a situation may occur in which output neurons have all been used and none can be allocated to encode a new sequence. To deal with this, we suggest the use of some unsupervised constructive algorithms [52] to include neurons when necessary. We believe that the model can be further improved and much work could be developed in the following directions:

1. Further comparison with approaches more recently proposed in the neural network literature, such as the models of Wang and Yuwono [33] and Srinivasa and Ahuja [53], so that the viability of unsupervised neural networks for spatiotemporal processing and its application in robot control can be firmly demonstrated.
2. A self-organizing mechanism to determine the depth of the time-varying context based solely on the input patterns. See, for example, Wang and Arbib [7] and Wang [21].
3. Implementation in a real robot: despite the many properties of the proposed neural system that can be inferred from computer simulations, the ultimate goal of a neural controller must be its testing in real robotic system. This is one of our next steps.

Acknowledgments

The authors would like to thank the following Brazilian research agencies for their financial support to this research: PICDT/CAPES/UFC and FAPESP.

References

1. A.Y. Zomaya and T.M. Nabhan, "Trends in neuroadaptive control for robot manipulators," in *Handbook of Design, Manufacturing and Automation*, edited by R.C. Dorf and A. Kusiak, John Wiley & Sons: New York, pp. 889–917, 1994.
2. O. Omidvar and P. van der Smagt (eds.), *Neural Systems for Robotics*, Academic Press: San Mateo, CA, 1997.
3. J.J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd edn., Addison-Wesley: Reading, MA, 1989.
4. P.C.Y. Chen, J.K. Mills, and K.C. Smith, "Performance improvement of robot continuous-path operation through iterative learning using neural networks," *Machine Learning*, vol. 23, pp. 75–104, 1996.
5. J. Heikkonen and P. Koikkalainen, "Self-organization and autonomous robots," in *Neural Systems for Robotics*, edited by O. Omidvar and P. van der Smagt, Academic Press: San Mateo, CA, pp. 297–337, 1997.
6. G. Bugmann, K.L. Koay, N. Barlow, M. Phillips, and D. Rodney, "Stable encoding of robot trajectories using normalised radial basis functions: Application to an autonomous wheelchair," in *Proceedings of the 29th International Symposium on Robotics (ISR'98)*, Birmingham, UK, 1998, pp. 232–235.
7. D.-L. Wang and M.A. Arbib, "Timing and chunking in processing temporal order," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 993–1009, 1993.
8. M.J. Denham and S.L. McCabe, "Robot control using temporal sequence learning," in *Proceedings of the World Congress on Neural Networks (WCNN'95)*, vol. II, Washington, DC, 1995, pp. 346–349.
9. T. Kohonen, *Self-Organizing Maps*, 2nd extended edn., Springer Series in Information Sciences, vol. 30, Berlin, Heidelberg, 1997.
10. A.F.R. Araújo and H.D'Arbo Jr., "Partially recurrent neural network to perform trajectory planning, inverse kinematics, and inverse dynamics," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 1998, pp. 1784–1789.
11. K. Althöfer and G. Bugmann, "Planning and learning goal-directed sequences of robot arm movements," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN'95)*, vol. 1, Paris, France, 1995, pp. 449–454.
12. R.N. Rao and O. Fuentes, "Learning navigational behaviors using a predictive sparse distributed memory," in *From Animals to Animats: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior*, Cape Cod, Massachusetts, 1996. MIT Press: Cambridge, MA, 1996, pp. 382–390.
13. S. Grossberg and M. Kuperstein, *Neural Dynamics of Adaptive Sensory-Motor Control*, Elsevier: Amsterdam, 1986.
14. M. Kuperstein and J. Rubistein, "Implementation of an adaptive neural controller for sensory-motor coordination," *IEEE Control Systems Magazine*, vol. 9, no. 3, pp. 25–30, 1989.
15. T.M. Martinetz, H.J. Ritter, and K.J. Schulten, "Three-dimensional neural net for learning visuomotor coordination of a robot arm," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 131–136, 1990.
16. H. Ritter, T. Martinetz, and K. Schulten, *Neural Computation and Self-Organizing Maps: An Introduction*. Addison-Wesley: Reading, MA, 1992.
17. D. Bullock and S. Grossberg, "Neural dynamics of planned arm movements: Emergent invariants and speed-accuracy properties during trajectory formation," *Psychological Review*, vol. 95, pp. 49–90, 1988.
18. J.A. Walter and K.J. Schulten, "Implementation of self-organizing networks for visuo-motor control of an industrial

robot," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 86–95, Jan. 1993.
19. D.-L. Wang and M.A. Arbib, "Complex temporal sequence learning based on short-term memory," *Proceedings of the IEEE*, vol. 78, pp. 1536–1543, 1990.
20. M.C. Mozer, "Neural net architectures for temporal sequence processing," in *Predicting the Future and Understanding the Past*, edited by A. Weigend and N. Gershenfeld, Addison-Wesley: Redwood City, CA, 1993, pp. 243–264.
21. D.-L. Wang, "Temporal pattern processing," in *The Handbook of Brain Theory and Neural Networks*, edited by M.A. Arbib, MIT Press, 1995, pp. 967–971.
22. G. de A. Barreto and A.F.R. Araújo, "Time in self-organizing maps: An overview of models," *International Journal of Computer Research*, in press. vol. 10, no. 2, pp 139-179, 2001.
23. A.V.M. Herz, "Spatiotemporal association in neural networks," in *The Handbook of Brain Theory and Neural Networks*, edited by M.A. Arbib, MIT Press: Cambridge, MA, 1995, pp. 902–905.
24. D.E. Rumelhart and J.L. McClelland (eds.), *Parallel Distributed Processing*, vol. 1, MIT Press: Cambridge, MA, 1986.
25. G.deA. Barreto and A.F.R. Araújo, "Fast learning of robot trajectories via unsupervised neural networks," in *Proceedings of the 14th IFAC World Congress*, Pergamon Press: Oxford, Beijing, China, 1999, pp. 373–378.
26. G. de A. Barreto and A.F.R. Araújo, "Unsupervised learning and recall of temporal sequences: An application to robotics," *International Journal of Neural Systems*, vol. 9, no. 3, pp. 235–242, 1999.
27. D.L. James and R. Miikkulainen, "A self-organizing feature map for sequences," in *Advances in Neural Processing Systems*, vol. 7, edited by G. Tesauro, D.S. Touretzky, and T.K. Leen, MIT Press: Cambridge, MA, 1995, pp. 577–584.
28. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley: Redwood City, CA, 1991.
29. D.O. Hebb, *The Organization of Behavior*, Wiley: New York, 1949.
30. S. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Transactions on Computers*, vol. C-21, no. 11, pp. 1197–1206, 1972.
31. P.I. Corke, "A robotics toolbox for MATLAB," *IEEE Robotics and Automation Magazine*, vol. 3, no. 1, pp. 24–32, 1996.
32. A.F.R. Araújo and M. Vieira, "Associative memory used for trajectory generation and inverse kinematics problem," in *Proceedings of the IEEE World Congress on Computational Intelligence, (WCCI'98)*, Anchorage, USA, 1998, pp. 2052–2057.
33. D.-L. Wang and B. Yuwono, "Incremental learning of complex temporal patterns," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1465–1481, 1996.
34. N.B. Toomarian and J. Barhen, "Learning a trajectory using adjoint functions and teacher forcing," *Neural Networks*, vol. 5, pp. 473–484, 1992.
35. D.-T. Lin, J.E. Dayhoff, and P.A. Ligomenides, "Trajectory production with the adaptive time-delay neural network," *Neural Networks*, vol. 8, no. 3, pp. 447–461, 1995.
36. D.-T. Lin, "Sampling effects on trajectory production and attractor prediction," *Journal of Information Science and Engineering*, vol. 13, no. 2, pp. 293–310, 1997.
37. P. van der Smagt, "Simderella: A robot simulator for neuro-controller design," *Neurocomputing*, vol. 6, no. 2, pp. 281–285, 1994.
38. B. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1212–1228, 1995.
39. B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.
40. M.J. Healy, T.P. Caudell, and S.D. Smith, "A neural architecture for pattern sequence verification through inferencing," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 9–20, 1993.
41. M. Hagiwara, "Time-delay ART for spatio-temporal patterns," *Neurocomputing*, vol. 6, no. 5/6, pp. 513–521, 1994.
42. S. Grossberg, "Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, I," *Journal of Mathematics and Mechanics*, vol. 19, pp. 53–91, 1969.
43. G.A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54–115, 1987.
44. S. Becker, "Implicit learning in 3D object recognition: The role of temporal context," *Neural Computation*, vol. 11, no. 2, pp. 347–374, 1999.
45. G. Tesauro, "Simple neural models of classical conditioning," *Biological Cybernetics*, vol. 55, no. 4, pp. 187–200, 1986.
46. P.R. Montague and T.J. Sejnowski, "The predictive brain: Temporal coincidence and temporal order in synaptic learning mechanisms," *Learning & Memory*, vol. 1, pp. 1–33, 1994.
47. G. Wallis, "Using spatio-temporal correlations to learn invariant object recognition," *Neural Networks*, vol. 9, no. 9, pp. 1513–1519, 1996.
48. B. Schölkopf and H. Mallot, "View-based cognitive mapping and path-planning," *Adaptive Behavior*, vol. 3, pp. 311–348, 1995.
49. M. Girolami and C. Fyfe, "A temporal model of linear anti-Hebbian learning," *Neural Processing Letters*, vol. 4, no. 3, pp. 139–148, 1996.
50. K. Kopecz, "Unsupervised learning of sequences on maps of lateral connectivity," in *Proceedings of the International Conference on Artificial Neural Networks*, Paris, 1995, pp. 431–436.
51. H. Hyötyniemi, "Locally controlled optimization of spray painting robot trajectories," in *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Istanbul, Turkey, 1990, pp. 283–287.
52. B. Fritzke, "Growing cell structures—a self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441–1460, 1994.
53. N. Srinivasa and N. Ahuja, "A topological and temporal correlator network for spatiotemporal pattern learning, recognition, and recall," *IEEE Transactions on Neural Networks*, vol. 10, no. 2, pp. 356–371, 1999.

Au: Pls. provide biography and photo