Context in Temporal Sequence Processing: A Self-Organizing Approach and Its Application to Robotics

Aluizio F. R. Araújo and Guilherme de A. Barreto, Student Member, IEEE

Abstract—A self-organizing neural network for learning and recall of complex temporal sequences is developed and applied to robot trajectory planning. We consider trajectories with both repeated and shared states. Both cases give rise to ambiguities during reproduction of stored trajectories which are resolved via temporal context information. Feedforward weights encode spatial information of the input trajectories, while the temporal order is learned by lateral weights through a time-delayed Hebbian learning rule. After training is completed, the network model operates in an anticipative fashion by always recalling the successor of the current input state. Redundancy in sequence representation improves the robustness of the network to noise and faults. The network uses memory resources efficiently by reusing neurons that have previously stored repeated/shared states. Simulations have been carried out to evaluate the performance of the network in terms of trajectory reproduction, convergence time and memory usage, tolerance to fault and noise, and sensitivity to trajectory sampling rate. The results show that the network model is fast, accurate, and robust. Its performance is discussed in comparison with other neural-network models.

Index Terms—Context, Hebbian learning, robotics, self-organization, temporal sequences, trajectory planning.

I. INTRODUCTION

A long-standing problem in neural-network theory is the mathematical and computational formalization of so-called spatio-temporal sequence learning [1], [2]. Processing of such sequences is a topic of research arising in several fields, such as dynamic pattern recognition in hearing and vision. Similarly, reproduction of such sequential patterns is important in processes like motor pattern production, speech, and singing. For these types of problem, neural-network architectures should provide two storage mechanisms: one for spatial information and the other for temporal information. The former mechanism is responsible for storing the sequence items without regard to their order of occurrence in time. The latter, called the short-term memory (STM) mechanism, must extract and store the temporal relationships between the items.

Trajectory planning is a robotic task in which the robot has to follow a prescribed path in time. It is currently programmed by an operator who guides the robot through a sequence of desired arm positions, whilst these positions are then stored in the controller memory for later recall. This method is time-consuming and uneconomical, since during the storage process the robot is out of productive activity [4] and the whole process occurs under the supervision of the robot operator. Furthermore, in some industrial settings, a robot is often required to perform more than one task and the robot controller must be able to follow more than one trajectory. As the task becomes more complex, the operator may experience difficulties in recording all the positions, specially if the robot has to follow multiple trajectories, passing several times through the same place.

Trajectory planning can be handled by neural networks for temporal sequence processing, if the trajectory to be followed is understood as a sequence of arm positions. The role of the neural network is to learn to associate consecutive states of a trajectory and store these state transitions for total or partial reproduction of the learned trajectory. During the reproduction, at the start of each cycle, the current position of the robot is presented to the network and it should respond with the next one, until a target position has been reached. Potential ambiguities during reproduction of multiple trajectories should be resolved, preferably by the network itself.

Most neural models for sequence processing and trajectory planning are supervised models based on standard MLP or dynamic recurrent networks [4]–[10]. They depend on the network designer to establish the correct temporal association between consecutive trajectory points, as well as to resolve ambiguities. In contrast, an unsupervised network can be used to learn the temporal order of a trajectory in an autonomous fashion. Likewise, uncertainties during reproduction can be resolved without the need of external help. Indeed, principles of self-organization of systems have proved to reduce substantially the burden of robot programming, responsible for about one-third of the overall costs of an industrial robotic system implementation [11]. Few self-organizing networks, however, have been proposed that are capable of reproducing robot trajectories [14]–[16]. Among them, only the network by Bugmann et al. [14] can handle trajectories with repeated points, while only the model by Barreto and Araújo [16] can deal with trajectories with shared points. None of them can learn and recall trajectories that not only contain repeated points but also share points with others.

Althöfer and Bugmann [13] described a neural implementation of a resistive grid for controlling a robotic arm during trajectory reproduction. This model produces “jerky” movements and an inaccurate final end-effector position, owing to the limited resolution of grid-based methods. Furthermore, it is unable to reproduce trajectories that contain repeated points or share
points with others. Bugmann et al. [14] then proposed a neural network which uses normalized radial basis functions (NRBF) [17] to learn the sequence of positions forming the trajectory of an autonomous wheelchair. After training is completed, the network responds to each input position by producing the next position for the wheelchair. As the trajectory may pass several times over a particular position, context information is added to the position information to avoid uncertainties caused by the aliasing problem (identical sensory inputs may require different actions from an autonomous system) [18]. This network learns trajectories with repeated points, but it is not able to learn multiple trajectories with points in common.

In the neural models just described, the temporal associations between successive robot positions are prewired, i.e., determined beforehand by the network designer. Barreto and Araújo [15], [16] proposed a self-organizing neural algorithm that learns the temporal order of an input trajectory in an unsupervised fashion. Individual trajectory points are stored in feedforward weights while their order of occurrence in time is encoded by lateral weights. With the help of fixed context input units, the network can learn multiple trajectories with points in common. This model, however, cannot handle trajectories with repeated points.

In this paper we propose a self-organizing network that overcomes the limitations of the neural models described in the previous paragraphs. It is able to learn and recall multiple trajectories with repeated and shared points, with the help of time-varying context information. The proposed network is evaluated in terms of its ability to reproduce accurately the stored trajectories and resolve ambiguities during trajectory reproduction, convergence time and memory usage, tolerance to fault and noise, and sensitivity to trajectory sampling rate. The results reported here show that the network model is fast, accurate, and robust. Unlike in the supervised approach, the ultimate goal in this work is to have the trajectory planning task automated with minimal human supervision.

The remainder of this paper is organized as follows. In Section II, we introduce the network topology and the corresponding learning algorithm. In Section III, computer simulations with the proposed model are performed in the context of a robotic application. In Section IV, we discuss the main features of the proposed network and its relation to other available models. We conclude the paper in Section V.

II. THE NEURAL NETWORK ALGORITHM

In this section, we introduce the mathematical formulation of the proposed model. The first thing to be defined is the nature (discrete or continuous) of the temporal dimension. Second, we describe the type of temporal sequence we are interested in and that the network is able to process. Details of the components of the network architecture are given in the following section.

For the purposes of simulation on digital computers, it is useful to discretize the temporal dimension by sampling at regular intervals and adopting a system in which time proceeds by intervals of $\Delta t$. We shall use the symbol $t$ to represent a particular point in time, where $t \in \{0, \Delta t, 2\Delta t, 3\Delta t, \ldots\}$. In this formulation, $\Delta t$ can be considered to be the unit of measure for the quantity $t$, and thus it is reasonable to omit the units and express $t$ simply as a member of the set of integer numbers, $t \in \{0,1,2,3,\ldots\}$.

A spatiotemporal sequence $S^l$ is a finite set of pattern vectors $s^l(t) = \{s^l_1(t), s^l_2(t), \ldots, s^l_M(t)\}$, $s^l(t) \in S \subset \mathbb{R}^n$, grouped according to their order of occurrence in time, that is, $S^l = \{s^l(t), s^l(t-1), \ldots, s^l(t-N^l+1)\}$ where $N^l$ is the length of the sequence $l$, $l = 1, \ldots, M$. The elements of $s^l(t)$ can be integer or real numbers. The feature vectors $s^l(t)$ can also be called sequence items, points, states, components, or elements.

We first classify sequences as open or closed. Open sequences are those in which the initial item is different from the final one. In closed sequences, the initial item is equal to the final one. Thus, the sequence of letters A-B-C-D-E is open, whereas X-Y-Z-W-X is closed. An open or closed sequence may have repeated items. For example, the sequences A-B-C-D-C-E and X-Y-Z-W-Z-X are examples of sequences with repeated items, respectively, open and closed. In addition, two or more sequences can have items in common (shared items). From now on, we refer to either repeated or shared elements of a sequence as recurrent. Sequences without recurrent items are called simple sequences, whereas those containing recurrent elements are referred to as complex. For instance, the sequences A-B-C-D-E and X-Y-C-W-Z are simple when considered separately, but complex when considered together, since they share the item C.

Complex sequences are responsible for ambiguities that occur during sequence reproduction, resolved only by the introduction of contextual information. It is worth emphasizing that the term context is used very generally here to mean any secondary or additional source of information, derived from a different sensory modality, a different input channel within the same modality, or the temporal history of the input.

The current model extends previous work on context and temporal sequence learning for trajectory planning [15], [16]. These previous architectures can handle only open temporal sequences with shared items, because they make use only of fixed-type context. This type of context is unable to deal with closed trajectories with repeated states. The solution is to include time-dependent 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.

A. The Network Architecture

The basic architecture of the proposed model is shown in Fig. 1. This two-layer network consists of a broadcasting input layer and a competitive output layer, which carries out the processing. The model has feedforward and lateral weights that play distinct roles in its dynamics. This architecture differs from those of standard competitive networks in possessing context units at the input and delay lines at the output. The delay lines, however, are needed only during training in order to learn temporal associations.

In robotics, variables of different types may be included in the sensory vector $s^l(t)$ that represents the state of the robot arm at a given instant in time. Among them, we are particularly concerned with the following: the spatial position of the
end-effector of the robot in a three-dimensional workspace, denoted by \( \mathbf{p}(t) = [x(t), y(t), z(t)]^T \), \( \mathbf{p}(t) \in \mathbb{R}^3 \); the joint angles associated with \( \mathbf{p}(t) \), represented by the vector \( \mathbf{\theta}(t) = [\theta_1(t), \theta_2(t), \cdots, \theta_k(t)]^T \), \( \mathbf{\theta}(t) \in \Theta \subset \mathbb{R}^k \), where \( k \) denotes the number of degrees of freedom of the robot arm; and the joint applied torques, \( \mathbf{\tau}(t) = [\tau_1(t), \tau_2(t), \cdots, \tau_k(t)]^T \), \( \mathbf{\tau}(t) \in \Gamma \subset \mathbb{R}^k \), which are responsible for the arm reaching the joint angles specified by \( \mathbf{\theta}(t) \). Angles are measured in radians and torques in N.m.

Hence, for the robotic task, the sensory input vector \( \mathbf{s}(t) \) at time step \( t \) is defined as \( \mathbf{s}(t) = [\mathbf{p}(t), \mathbf{\theta}(t), \mathbf{\tau}(t)]^T, \mathbf{s}(t) \in \mathbb{R}^{3+k} \). Each \( \mathbf{s}(t) \) defines the state of the robot arm at a given instant of time, or equivalently, it defines the arm posture at time \( t \). In this sense, we can understand a robotic trajectory as a spatiotemporal sequence of postures of the robot arm, in which the spatial portion is represented by the static postures at given time steps and the temporal portion by the order in which the postures occur.

Context units are of two types: fixed and time-varying. The fixed context is denoted by \( \mathbf{C}_F \), \( \mathbf{C}_F \in C \subset \mathbb{R}^n \). It is time-invariant and usually set to a particular state from the temporal sequence, the initial or final one being the usual option. It is kept unchanged until the end of the current sequence has been reached. The fixed context acts as a kind of global sequence identifier.

Time-varying context is a kind of STM which can be implemented in various ways. For convenience, we adopted the simplest STM model, the so-called tapped delay lines [5], [19]. It consists of a sliding “time window” over the input sequence, collecting the corresponding samples and concatenating them, successively, into a single pattern vector of higher dimensionality. Time dependence between successive samples is then captured by the order of the concatenated elements in the input vector. Thus, time-varying context is the temporal history of the inputs, formed by the concatenation of past sequence items, \( \mathbf{C}_T(t, L) = [\mathbf{s}(t-1), \cdots, \mathbf{s}(t-L)] \), \( \mathbf{C}_T(t, L) \in \mathbb{R}^n \circledast \cdots \circledast \mathbb{R}^n \subset \mathbb{R}^{nL} \) where \( L \) is an integer called the memory depth [5]. A suitable length of the time window is usually determined after some experimentation with the data and the network.

The sensory input, the fixed and the time-varying context are then put together to form the total input pattern \( \mathbf{v}(t) = [\mathbf{s}(t), \mathbf{C}_F, \mathbf{C}_T(t, L)] \), \( \mathbf{v}(t) \in \mathbb{R}^{3+n+2k} \circledast \mathbb{R}^n \circledast \mathbb{R}^{nL} \subset \mathbb{R}^{3+n+2k+n(n+1)} \), to be presented to the network at time \( t \).

A trained network should provide information about the sequence item currently at the input and its successor in the sequence. In order to do so, four quantities are associated with an output neuron \( j \): 1) the activation value, \( a_j(t) \), responsible for identifying the sequence item currently at the network input; 2) the output value, \( y_j(t) \), responsible for identifying the next sequence item; 3) the feedforward weight vector, \( \mathbf{w}_j(t) \), connecting the input units to the output neuron \( j \); and 4) the lateral weight vector, \( \mathbf{m}_j(t) \), connecting the output neurons to the output neuron \( j \). These quantities and their role in the network dynamics during learning and testing are detailed next.

B. Learning the Static Arm Postures

The feedforward weight vector \( \mathbf{w}_j(t) = [\mathbf{w}_j^1(t), \mathbf{w}_j^2(t), \mathbf{w}_j^3(t)] \), \( \mathbf{w}_j(t) \in \mathbb{R}^{3+2k} \circledast \mathbb{R}^n \circledast \mathbb{R}^{nL} \subset \mathbb{R}^{3+2k+n(n+1)} \), connects the input units to the output neuron \( j, j = 1, \cdots, m \). Each weight vector encodes an item of a particular sequence through a competitive learning rule. That is, for a particular sequence item, a single neuron (or a small group of output neurons) is responsible for storing such an item. The input state is compared with each feedforward weight vector in terms of Euclidean distance. Thus, we define a sensory distance, \( D_j^s(t) \in \mathbb{R} \), a fixed context distance, \( D_j^f(t) \in \mathbb{R} \), and a time-varying context distance, \( D_j^T(t) \in \mathbb{R} \), as follows:

\[
D_j^s(t) = ||\mathbf{s}(t) - \mathbf{w}_j^s(t)||, \quad D_j^f(t) = ||\mathbf{C}_F(t) - \mathbf{w}_j^F(t)|| \quad \text{and}
\]

\[
D_j^T(t) = ||\mathbf{C}_T(t, L) - \mathbf{w}_j^T(t)||. \quad (1)
\]

The distance \( D_j^s(t) \) is used to find the winners of the current competition, while \( D_j^f(t) \) and \( D_j^T(t) \) are used to solve ambiguities during recall.

For accuracy in planning, every item of a sequence should be memorized for posterior recall in the correct temporal order. That is, if the input sequence \( h \) has \( n \) items, all \( n \) items should be stored and recalled in the correct order. Standard competitive networks, however, tend to cluster the input patterns by similarity and may split the trajectory in discontinuous segments, causing a jerky movement of the robot arm. To overcome this situation, the network “penalizes” each neuron, by excluding it from subsequent competitions, once it has been chosen to be a winner. To this end, we define a function \( R_j(t) \), termed the responsibility function, that indicates if a neuron \( j \) is already responsible for the storage of a given item of the sequence. If \( R_j(t) > 0 \), neuron \( j \) is excluded from subsequent competitions for sequence components. If \( R_j(t) = 0 \), neuron \( j \) is allowed to compete.

In order to use memory space efficiently, every time a recurrent item occurs, it should be encoded by the same neuron that encoded it the first time, otherwise many copies of this item will exist in the network as in the model by [16]. This is accomplished by defining a similarity radius, \( 0 < \varepsilon \leq 1 \), that accounts for the repetition of sequence items. If neuron \( j \) has never won a competition (i.e., \( R_j(t) = 0 \)) or if the pattern stored in its weight vector is within the neighborhood of the current input (i.e., \( D_j^s(t) \leq \varepsilon \)), then neuron \( j \) will be evaluated, for the purpose of competition, by its sensory distance \( D_j^s(t) \) only. Otherwise, this distance is weighted by the responsibility function, ex-
cluding neuron $j$ from subsequent competitions. This behavior can be formalized in terms of a function $f_j(t)$, called choice function, as follows:

$$f_j(t) = \begin{cases} D_j(t) \text{ if } D_j(t) \leq \varepsilon \text{ or } R_j(t) = 0, \\ R_j(t) \text{ otherwise.} \end{cases} \tag{2}$$

In summary, for the purpose of training, a high value for the similarity radius (i.e., $\varepsilon \approx 1$) would cluster the states of the current input, i.e., $(D_j(t) \approx 0 \leq \varepsilon)$. For the purpose of trajectory reproduction, the similarity radius may assume higher values ($\varepsilon \approx 1$), to avoid incorrect evaluation of (2) caused by distorted input, resulting from noisy measurements of the sensory vector $s(t)$.

The output neurons are then ranked according to the values of their choice functions as follows:

$$f_{\mu_i}(t) < f_{\mu_2}(t) \ldots < f_{\mu_m}(t) \tag{3}$$

where $m$ is the number of output neurons and $\mu_i(t), i = 1, \ldots, m$, is the index of the $i$th neuron closest to the current sensory vector $s(t)$. We consider $K$ neurons, $\mu(t) = [\mu_1(t) \mu_2(t) \cdots \mu_K(t)]^T$, $K \leq m$, as winners of the current competition. They will represent the current input vector $\nu(t)$. The integer $K$ is called the redundancy parameter: it specifies the number of neurons that will store each trajectory state.

The activation values decay linearly from a maximum value $\alpha_{\text{max}} \in \mathbb{R}$ for $\mu_1(t)$, to a minimum $\alpha_{\text{min}} \in \mathbb{R}$ for $\mu_K(t)$, as follows:

$$\alpha_{\mu_i} = \begin{cases} \alpha_{\text{max}} - \left( \frac{\alpha_{\text{max}} - \alpha_{\text{min}}}{K-1} \right) (i-1), & \text{for } i \leq K \\ 0, & \text{for } i > K \end{cases} \tag{4}$$

where $\alpha_{\text{max}}$ and $\alpha_{\text{min}}$ are user-defined. Other monotonically decreasing functions are equally possible for the activations.

The responsibility function $R_j(t)$ is updated every time a new activation pattern $a(t) = [a_1(t) \cdots a_m(t)]^T, a(t) \in A \subset \mathbb{R}^m$, is computed

$$R_j(t+1) = R_j(t) + \beta a_j(t) \tag{5}$$

where $\beta > 1$ is termed the exclusion parameter. Next, the parts forming the weight vectors $w_j(t)$ are adjusted as follows:

$$w_j(t+1) = w_j(t) + \eta a_j(t) [s(t) - w_j(t)] \tag{6}$$

$$w_i(t+1) = w_i(t) + \eta a_i(t) [C_i(t) - w_i(t)] \tag{7}$$

$$w_{ij}(t+1) = w_{ij}(t) + \eta a_j(t) [C_i(t) - w_{ij}(t)] \tag{8}$$

where $0 < \eta \leq 1$ is the feedforward learning rate. For $t = 0$, $w_j(0)$ is initialized with random numbers between zero and one. The parameter $\eta$ controls the overall speed of learning.

The net effect of applying (6)–(8) is that the trajectory state currently at the network input is stored in the weight vectors $w_{\mu_i}(t)$ of the $K$ first winners, because only they have nonzero activations, i.e., $a_{\mu_i} \neq 0$ for $i \leq K$. The fixed and time-varying context associated with the current trajectory state are also stored.

It is worth noting that the state trajectory and its contexts are not stored with the same intensity in the $K$ neurons. The degree of intensity (or similarity) is determined by the position of the neuron in the ranking shown in (3) which is reflected in the activation pattern in (4).

C. On the Convergence and Stability of Feedforward Learning

For the proposed model to be useful in practice, learning must be fast. With a large learning rate (i.e., $\eta \approx 1$), the network exhibits so-called one-shot learning: storing a trajectory state on a network after the first presentation. One-shot learning has previously been used in the ART models [20] and constitutes an important feature of the neural network being described in this paper.

As an example, let us adjust the weights of the first winner, $\mu_1$. In this case, $a_{\mu_1} = 1$. After rearranging the terms, the learning rule in (6) becomes

$$w_{\mu_1}(t+1) = (1-\eta)w_{\mu_1}(t) + \eta s(t).$$

This equation tells us that the weight $w_{\mu_1}$ is modified such that its previous value is retained by a factor of $(1-\eta)$, while the current trajectory state $s(t)$ affects the weight by a factor of $\eta$.

If we assume a value of $\eta = 1$, then we get

$$w_{\mu_1}(t+1) = s(t).$$

where the current state is directly stored in the weights. Thus, the entire trajectory is stored, state-by-state, in a single pass of its states. In other words, there is no need to present a trajectory to the network more than once. In this sense, convergence time is substantially reduced, compared to that of a supervised network.

In many situations, however, one-shot learning may lead to problems of memory storage and retrieval [21]. The proposed network saves memory space by employing the responsibility function and the corresponding similarity radius, which ensure that only a single copy of a recurrent state is stored. Furthermore, one-shot learning combined with the exclusion mechanism allows stable learning of multiple trajectories, in the sense that acquisition of new trajectories does not interfere with previously stored ones.

D. Learning the Temporal Order of Stored Arm Postures

One of the goals of our model is to explore a simple Hebbian mechanism for processing temporal order. The simple premise is that connections between neurons depend not only upon current input and current neural activity, as Hebb proposed [22], but maybe upon previous neural activity as well. It has been recognized that time-delayed Hebbian learning rules play an essential role in classical conditioning [23], [24], object recognition [25], route learning and navigation [26] and blind source separation [27]. To our knowledge, the model proposed here, together with those in Barreto and Araújo [16], [15] are the first to use a time-delayed Hebbian rule to learn the temporal order of a sequence and apply it to trajectory planning of a robotic manipulator.

The set of lateral weights $m_j(t) = [m_{j1} m_{j2} \cdots m_{jm}]^T$, $m_j(t) \in \mathbb{R}^m$ stores the temporal order of the patterns in a sequence, using a Hebbian learning rule to associate the winner of
to the previous competition with the winner of the current competition. The lateral weights are updated according to the following learning rule:

\[
m_{jr}(t+1) = \begin{cases} 
    m_{jr}(t) + \lambda a_{jr}(t) a_r(t-1) & \text{if } m_{jr}(t) \neq 0 \\
    m_{jr}(t) & \text{otherwise}
\end{cases}
\]  

(9)

where \(0 < \lambda < 1\) is a gain parameter. Equation (9) can be understood as a Hebbian-like rule that learns temporal associations between pairs of winning neurons that encoded consecutive items of the input sequence. The network initially has no lateral connections, i.e., \(m_{jr}(0) = 0\) for all \(j, r\), indicating that no temporal associations exist at the beginning of the training. As training proceeds, the network establishes a lateral connection from the winner at time \(t\) to the winner at time \(t\). In Fig. 2, at \(t = 1\) the neuron on the left is the winner for pattern \(s(1)\). This pattern is copied to the feedforward weight vector of the winner, i.e., \(w^{a}_{j_1}(1) \leftarrow s(1)\). At \(t = 2\), the neuron on the right is the winner for pattern \(s(2)\) and its feedforward weight vector is adjusted accordingly, i.e., \(w^{a}_{j_2}(2) \leftarrow s(2)\). Still at \(t = 2\), a lateral connection is created from the neuron on the left to the neuron on the right according to (9). This is equivalent to saying that the network "looks" backward one time step to establish a causal link corresponding to the temporal transition between two consecutive sequence items, \(s(t-1) \rightarrow s(t)\). This transition is encoded in the lateral weight connecting the neurons associated with the activation pairs \([a_{j_1}(t-1), a_{j_2}(t)]\), \(i \leq K\). Successive application of (9) eventually leads to the encoding of the temporal order of the sequence.

It is worth noting that a lateral connection \(m_{jr}(t)\), encoding a single temporal transition, is learned only once, even if it happens to occur again in the future. This is necessary for weight normalization and to avoid the creation of biased transitions. If a transition occurs several times within a sequence, the incremental adjustment of the corresponding lateral weight would eventually force this connection to assume high values. A strong lateral connection can bias the recall process by favoring the transition it encodes, even if the context information suggests the use of another transition (see Subsection II.D for details of the recall process).

E. Recall of a Stored Trajectory

Once a trajectory is learned, it can be retrieved either from its initial or any intermediate state. It should be noted that the trajectory recall process is a closed-loop control scheme in which the robot is inserted (Fig. 3). The recall (or reproduction) process comprises five steps: 1) recall initiation; 2) computation of neuronal activations; 3) computation of neuronal outputs; 4) delivery of control signals to the robot; and 5) determination of sensory inputs. For recall, the parameter \(K\) is always set to one.

1) Recall Initiation: To initiate reproduction, any trajectory state can be presented to the network by the robot operator \((t = 0\) in Fig. 3). For \(t > 0\), the network dynamics will then evolve autonomously to recall the part of the stored trajectory that follows that triggering state.

2) Computation of Activations: For each input state, the activation \(a_{t}^{\mu}\) of the winning neuron, \(\mu\), should be computed according to (4). The feedforward weight vector \(w^{a}_{\mu}(t)\) of the winner is the closest to the current input state.

3) Computation of Outputs: The winner—the only output neuron with \(a_{t}^{\mu} > 0\)—will then trigger the neuron whose weight vector corresponds to the successor of the current input state in the stored trajectory. This is possible because of the state transition learned during the training phase and encoded by a lateral weight connecting these neurons. Thus, the output equation is as follows:

\[
y_{j}(t) = \left(1 - \frac{D^F_{j}(t)}{\sum_{r=1}^{m} D^F_{r}(t)}\right) \cdot \left(1 - \frac{D^J_{j}(t)}{\sum_{r=1}^{m} D^J_{r}(t)}\right) \cdot G \left(\sum_{r=1}^{m} m_{jr}(t) a_r(t)\right)
\]

(10)

where \(G(u) \geq 0\) and \((dG(u)/du) > 0\).

In the case of a trajectory with no shared or repeated elements, the third factor on the right-hand side of (10) will correctly indicate the neuron that stored the next state of the trajectory. In the case of a trajectory with recurrent states, additional disambiguating information is required, because the third factor alone will produce the same value of \(y_{j}(t)\) for all candidates for the next item of the sequence. The ambiguity is resolved by the first and second factors on the right-hand side of (10): the candidate neuron whose stored fixed and time-varying context produce the highest values for the first and second factors,
i.e., the lowest values for $D_T(t)$ and $D_F(t)$, respectively, is considered the correct one to be chosen. This disambiguating ability will be further elucidated through a simulation example in Section III.

4) Delivery of Control Signals: The weight vector of the neuron with highest value of $y_j(t)$ contains the next trajectory state, i.e.,

$$u_{ctrl}(t) = w_{j^*}(t) \equiv \begin{pmatrix} p_{next}^* \\ q_{next}^* \end{pmatrix}$$

where $j^* = \arg \max_j [y_j(t)]$. The vector $u_{ctrl}(t)$ is then used to control the robot arm.

5) Determination of Sensory Inputs: When the robot arm attains its new position, a new sensory vector $s$ is formed with current sensor readings and fed back in order to be presented to the network. Steps 2)–5) continue until the end of the stored sequence. For $t = 0$, the activation and output values are set to $a_j(0) = y_j(0) = 0$, for all $j$.

It is worth noting that, during the recall process, if the network receives as input a state belonging to the stored trajectory, it searches for another stored state, such that the latter forms a state transition together with the input state. This is equivalent to saying that the network “looks” forward one time step, in order to output the stored pattern that succeeds in time the current one. In summary, during learning the network has a past-oriented behavior, while during recall the network has a future-oriented (anticipative) behavior.

F. A Hypothetical Example

A hypothetical example can be constructed, based on the concept of temporal associative memory developed by Amari [2], to elucidate the temporal order learning and one-step-ahead recall. Assuming that no recurrent item is present in the sequence to be learned, we can write (9) in matrix form as follows:

$$M(t + 1) = M(t) + \lambda a(t)a^T(t - 1).$$

For a sequence $I$ with $N$ items, we assume that the resulting matrix is given by a linear superposition of all contributions

$$M = M(0) + \lambda \sum_{t=1}^{N-1} a(t)a^T(t - 1).$$

Note that this matrix is constructed in an incremental and unsupervised way. It cannot be set in advance as in other associative memory models [28], [29], since the activation patterns $a(t)$ are not known beforehand.

Consider a trajectory with only three nonrepeated states and a network with three neurons. For the sake of simplicity, we set $K = 1$, and 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 $a(1) = [1 \ 0 \ 0]^T$, $a(2) = [0 \ 0 \ 1]^T$ and $a(3) = [0 \ 1 \ 0]^T$. Thus, in accordance with (13), the learned lateral memory matrix is

$$M = M(0) + \lambda \{a(3) a_T(2) + a(2) a_T(1) + a(1) a_T(0)\}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \lambda \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix}. \quad (14)$$

To illustrate how the matrix $M$ constructed by (14) retrieves the next sequence item, consider the following function: $G(u) = u$, for $u \geq 0$ and $G(u) = 0$, otherwise. Since there are no repeated elements to cause ambiguities, the first two factors on the right-hand side of (10), corresponding to the distances $D_T(t)$ and $D_F(t)$, are always equal to one. Also, note that $\sum_{i=1}^{N} w_{ij} a_i \geq 0$ in (10). Then we have a linear relationship for recall: $y(t) = Ma(t)$.

If the first item in the sequence, $s(1)$, is presented to the network, the resulting activation and output patterns are as follows:

$$a(1) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \rightarrow y(1) = Ma(1)$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \lambda \end{pmatrix}$$

which indicates that neuron $j^* = \arg \max_j [y_j(1)] = 3$ stored the second 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, i.e., $u_{ctrl}(1) \equiv w_3^*$. Once a robot has reached its next position, the new state of the robot arm is read by the sensors. These sensor readings are then fed back to the neural-network input to form the sensory vector $s(2)$. The presentation of $s(2)$ to the network, together with context information, results in the following activation and output patterns:

$$a(2) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \rightarrow y(2) = Ma(2)$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \lambda \end{pmatrix}$$

which indicates that neuron $j^* = 2$ stored the last trajectory state in its weight vector. The control signal is then $u_{ctrl}(2) = w_2^*$. When the robot arm reaches its final position, new sensor readings form the sensory vector $s(3)$. Together with context
information, the presentation of \( s(3) \) results in the following activation and output patterns:

\[
a(3) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow y(3) = Ma(3)
\]

\[
= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \lambda \\ \lambda & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
\]

which indicates that the trajectory has reached its end, because there is no neuron whose weight vector stored the "next state."

### G. Learning Input–Output Mappings

Due to the heterogeneous variables in the input sensory vector \( s'(t) \) in the robotic task under consideration, we can build, after learning, a variety of sensorimotor mappings through the use of a projection matrix \( P \) \[30], \[31\]. This is a diagonal matrix, whose elements are 0’s or 1’s, depending on the type of input-output mapping required. For example, if only the vector \( p(t) \) is given, the sensory vector is as follows:

\[
s'(t) = [p(t) \ 0 \ \cdots \ 0]^T
\]

where the elements of the vectors \( \theta(t) \) and \( \tau(t) \) have been made equal zero. Indeed, it does not matter what values \( \theta(t) \) and \( \tau(t) \) assume, since they will not contribute to the search for a winner. Thus, it is possible to search for the winner based only on \( p(t) \) by redefining \( D'_j(t) \) as follows:

\[
D'_j(t) = (s'(t) - w^s_j(t))^T P (s'(t) - w^s_j(t)) \quad (15)
\]

where the matrix \( P \) is chosen as

\[
P = \begin{pmatrix} I_p & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_p \end{pmatrix}
\]

where \( I_p \) is an identity matrix with \( \dim(I_p) = \dim(p(t)) \times \dim(p(t)) \). If we define \( \bar{P} = I_p - P \) as the complement of \( P \), where \( I_p \) is an identity matrix with the same dimension as \( s'(t) \), the control vector \( u_{ctrl} \) is now computed as follows:

\[
u_{ctrl}(t) = \bar{P}w^s_j(t) = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_p \end{pmatrix} \begin{pmatrix} p^{\text{ext}} \\ \theta^{\text{ext}} \\ \tau^{\text{ext}} \end{pmatrix}
\]

where \( j^* = \arg \min_j \| y_j(t) - \hat{y}(t) \| \), and \( I_\theta \) and \( I_\tau \) are identity matrices both of dimension \( \text{dof} \times \text{dof} \). The control vector \( u_{ctrl}(t) \) outputs the joint angles and joint-applied torques corresponding to \( P_{next} \). In this sense, one can say that the network is performing a kind of anticipative inverse kinematic and inverse dynamic computation.

It is worth noting that the distance \( D'_j(t) \) as defined in (15) can be changed on demand, which allows the user to choose freely the desired input-output mapping. One can, for example, reverse the mapping direction or switch to other input coordinate systems, using the same network topology. In the next section we present simulations with the proposed model engaged in trajectory planning tasks.

### III. Simulations

The trajectories used to train the network were generated by the toolbox \texttt{ROBOTICS} of Matlab [32] using the arm parameters of a PUMA 560 robot with six degrees of freedom. Notwithstanding the fact that we have chosen this robot for the simulations, the neural-network model is manipulator-independent. As long as a number of trajectories is available, learning can take place and does not depend on the robot arm parameters. Mathematical relationships among them are learned implicitly when the input sensory (state) vector \( s'(t) \) is formed and stored in the network. As described in Section II-G, by manipulating a projection matrix it is possible to learn the underlying nonlinear mappings associated with the robotic task at hand. This is an associative mechanism that forms the basis of many unsupervised neural-network models applied to robotic tasks, and has been used extensively to learn the forward and inverse kinematics and dynamics of a given manipulator [31], [33]–[35].

Four trajectories were considered here. The first three consist of open trajectories that share the same final state \( s_{\text{final}} \). In which the final spatial position is \( p_{\text{final}} = [0.6, 0.1, 0.0]^T \). Each trajectory has \( N_t = 10 \) states including the initial and final ones. The fourth trajectory is a closed trajectory (figure-eight sequence) with a repeated element at the spatial position \( p = [0.5, 0.5, 0.5]^T \). This simple trajectory is often used as a benchmark for sequence learning and reproduction [36], [37], because it emphasizes the need for time-varying contextual information during processing of sequences with repeated items. More recently, it has been used in robotics [38]. In this paper, figure-eight trajectories will also be used to evaluate the sensitivity of the network to the trajectory sampling rate, i.e., the number of discrete points forming the trajectory.

We are going to evaluate the ability of the network to learn and retrieve noise-free trajectories, and its robustness to noise and faults. The network performance will be evaluated in planning tasks by means of the mean square error (MSE), given by the following equation:

\[
\text{MSE}_t = \frac{1}{N_t} \sum_{t=1}^{N_t} \| p^t(t) - p^{\text{d}}(t) \|^2
\]

where \( N_t \) is the number of states in trajectory \( p_t \), \( p^t(t) \) and \( p^{\text{d}}(t) \) are the recalled and desired spatial coordinates of the robot end-effector. For training purposes, the trajectories were presented to the network one after the other.

1) **Learning and Recall of Noise-Free Open Trajectories:** In the first experiment, the three open trajectories were presented sequentially to the network. For the time being, we assume a noise-free sequence presentation, i.e., absence of noise in the
sensor measurements. Fig. 4 illustrates the learning (upper row) and recall (lower row) of one of the open trajectories in a simulated robot workspace.

A network with \( m = 10 \) neurons was used in this test. The other parameters were \( K = 1, \varepsilon = 10^{-5}, \alpha_{\text{defl}} = 6, L = 2, \sigma_{\text{max}} = 1, \beta = 1000, \eta = 1, \) and \( \lambda = 1. \) There was no need to specify \( \sigma_{\text{min}} \) because \( K = 1. \) The fixed context was set to \( C_F = \mathbf{p}_{\text{final}} \) whilst the time-varying context was set to \( C_T(t, 2) = (\mathbf{p}(t-1), \mathbf{p}(t-2)). \) For \( t = 0, C_T(0, 2) = \{0, 0\}. \) The upper row in the figure shows the neuron index and the arm posture encoded by the network during training. The lower row shows the sequence recall order (from left to right) of the output neurons during the performance phase. That is, neuron 2 retrieved the first state of the trajectory, neuron 8 the second, and so forth, until the last posture was recalled by neuron 7. Thus, the postures forming the robot arm trajectory were retrieved in the correct temporal order. The entire trajectory and the other two are shown in Fig. 5. The corresponding MSE is zero, indicating perfect learning and recall of the three trajectories, because we set \( \eta = 1. \)

2) Convergence Time and Memory Usage: The same set of open trajectories was previously used to train recurrent neural networks [9]. For comparison, the convergence time for each of the networks written in C and running on a PC of 200 Mhz was about 5s for the network proposed in this paper and 1 h for the recurrent neural network (for a final MSE error of 1%). The recurrent network had 15 neurons (three for spatial positions, six for joint angles, and six for joint torques) in the input and output layers, and a hidden layer with 20 neurons, and it was trained with standard backpropagation with adaptive learning rate and momentum. It is worth mentioning that this recurrent network, in contrast to the self-organizing one, is unable to start the reproduction process from any intermediate position.

The three open trajectories were also previously used to evaluate earlier developments of the model presented in this paper. To store the 30 states, the network in Barreto and Araújo [15], [16] requires 30 neurons, while the current model requires 28, since the final state is stored only once. The larger the number of occurrences of a given point of the trajectory, the larger the savings in memory use. Both models possess the same ability regarding the fast and accurate learning and recall of open trajectories. In addition, the current network includes a time-varying context which allows it to process closed trajectories with repeated states (See next simulation).

3) Learning and Recall of Closed Trajectories: The next simulation evaluates the ability of the network to learn and retrieve a figure-eight sequence with 49 items. Unlike open trajectories, a target position is not defined in a closed one. In this case, the fixed context is set to the first sequence item presented to the network, i.e., \( C_F = \mathbf{s}(1). \) The training parameters were the same as in the last simulation, except for \( K = 5, \sigma_{\text{min}} = 0.05 \) and the number of neurons \( m = 150. \) The result of the training and recall is shown in Fig. 6.

It worth noting that this simulation also illustrates the fault-tolerance of the network due to \( K > 1. \) That is, in case of failure of all the first winners \( \mu_1(t) \) for a given trajectory, the second winners \( \mu_2(t) \) will then be responsible for the recall of this trajectory, with a slightly higher error \( \text{MSE}_{\mu_2}^{\text{tar}} < 0.000058 \). If all the second winners \( \mu_2(t) \) collapse, the recall is executed by the third winners \( \mu_3(t) \), with a higher error \( \text{MSE}_{\mu_3}^{\text{tar}} < 0.000068 \), and so forth.

Intermediate values for recall errors occur in the case of failure of a single neuron or small group. For example, if only five out of 49 of the first winners collapse, the corresponding second winners and the other 44 first winners will recall the required trajectory with an error \( 0 < \text{MSE}_{\mu_1} < 0.000058. \) The same behavior is observed for the corresponding joint angles and torques. Fig. 7 shows the elbow angles for each trajectory in Fig. 6.

Fig. 8 illustrates how (10) resolves ambiguities. First of all, it is important to note that, no matter what branch of the figure-eight the robot arm is currently executing, when it arrives at the crossing point, it has to decide between one of two possible directions to follow. This ambiguity is resolved by using the time-varying context.

In the figure-eight trajectory, the state \( \mathbf{s}_{212} \) is equal to state \( \mathbf{s}_{36} \), but they occur in different contexts. During training these states were encoded by neuron 23 (see Fig. 8). During recall, when the robot arm reaches the posture corresponding to state \( \mathbf{s}_{212} \), the network has to decide which direction to follow in outputting the next robot state. Following the arrows representing the lateral connections in Fig. 8, the candidates for the next state are stored in the weight vectors of neurons 3 and 71. Since the trajectory is closed, the fixed context is the same for the two neurons. Only the time-varying context contains the information that can resolve the ambiguity (i.e., the past spatial positions \( \mathbf{p}_{10} \) and \( \mathbf{p}_{11} \) in \( \mathbf{w}_{\mathbf{s}} \) and \( \mathbf{C}_T(t, 2). \)) Thus, \( D_{\mathbf{p}_{10}}^T(t) < D_{\mathbf{p}_{11}}^T(t) \), implying that \( y_3 > y_1. \)

So far, the simulations have assumed noise-free trajectories. Now, the network performance under noisy conditions will be evaluated. We consider a zero-mean Gaussian noise, with variance \( \sigma^2 \in [0, 0.2]. \) Fig. 9 shows a typical noise-tolerance test and its relation to the value given to the redundancy number \( K. \) It can be inferred from this figure that for \( 1 \leq K < 3, \) the...
no noise sensitivity does not change significantly. For \( K \geq 3 \), the error curves clearly follow a linear trend with increasing variance. This occurs because high values of \( K \) increase the chance of superposition between consecutive stored states, resulting in higher errors. This behavior was observed in other types of trajectory, suggesting that values of \( K \geq 3 \) demand high memory resources and result in high sensitivity to noise.

The next experiment evaluates the network performance for the same figure-eight trajectory with different sampling rates, i.e., different numbers of trajectory states. We trained the net-
work on trajectories with 25 and 97 states, which correspond to approximately half and double the sampling rate of the trajectory in Fig. 6. The parameters were the same as in the former simulation, and the results are shown in Fig. 10.

The error curves follow the same behavior as those in Fig. 9 with respect to the influence of $K$. However, two points should be emphasized with respect to the influence of the sampling rate: 1) higher sampling rate demands a higher memory usage, but produces lower recall errors and 2) the mean and variance of the error curves increase as the sampling rate decreases (see Table I).

In summary, for $K \geq 3$, the memory requirements of the network and the recall errors increase. Moreover, augmentation of the sampling rate causes a rise in memory demands, but a fall in recall errors.

The sampling rate depends on the task to be performed and is usually defined in advance by the robot operator. For the robotic task we are interested in, there is usually enough memory space to accommodate the manipulator trajectories to be followed with the predefined sampling rates. The algorithm proposed here was developed for this case, for which it is guaranteed to have very good performance. In Section IV we show how to roughly determine the number of output neurons when the number of input trajectories and their sampling rates are known. The situation may change for other robotic tasks such as mobile robot navigation. In this case the sampling rates during exploration of the environment are usually very high and the memory space is limited. Hence, the sampling rate should be sufficiently high to reproduce the stored trajectories as faithful as possible, but sufficiently low to fit the memory capacity of the mobile robot controller. In Section V we briefly discuss how to change the algorithm proposed here to handle such a situation.

### IV. DISCUSSION

The proposed model makes use of concepts coming from fields as diverse as neuroscience, psychology, and robotics. The result is a self-organizing network with appealing characteristics that make it suitable for applications requiring simple, fast and accurate learning and recall of complex temporal sequences. In addition, its formulation gives rise to important aspects of neural-network learning that we discuss next.

1) **The Look-Up Table Approach:** One of the earliest connectionist approaches to robot control is due to Albus [39]. His cerebellar model articulation controller (CMAC) has no prior knowledge of the structure of the system being controlled and thus could be trained to accomplish the robot control task. The basic idea of CMAC is to compute control commands by look-up tables rather than by solving control equations analytically. The table is organized in a distributed fashion, so that the function value for any point in the input space is derived by combining the contents of a number of memory locations. The model proposed here works much like a look-up table, since an input trajectory state is associated with an output neuron, with further information available in its associated feedforward and lateral weights. Such information consists of control variables such as the joint angles and torques. Furthermore, the proposed model is unsupervised and offers some degree of tolerance to noise and faults, issues not easily addressed in conventional approaches to robot control via the lookup table method. Finally, the proposed model handles ambiguous situations as easily as unambiguous ones, implying that it can be scaled up to deal with more complex trajectories without additional difficulty.

2) **Associative Chaining and Temporal Context:** In his seminal paper, Lashley [40] rejected simple associative chaining as a plausible hypothesis for dealing with complicated aspects of temporal order of behavior. However, in recent years, a considerable number of neural models based on simple associative chaining have been proposed, showing that simple assumptions can be made as long as contextual information is provided to resolve potential ambiguities. The majority of the models for sequence processing are based on either multilayer perceptrons (MLPs) with some temporal version of backpropagation training [7] or the Hopfield model of associative memory [28], [6], [41]. Also, BAM-type [29], [42] and ART-type [43], [44] models use the simple associative chaining hypothesis, and have been applied to a variety of complex tasks in natural language processing, time-series analysis, and motor control. The model proposed in this paper also follows this paradigm. However, when compared with the models based on MLP and BAM, the proposed one learns the temporal association in a self-organized manner and the learning process is much faster.
Fig. 10. The combined influence of the redundancy parameter $K$ and the sampling-rate of the sequence on the noise-tolerance of the network. (a) Figure-eight trajectory with 25 states. (b) Figure-eight trajectory with 97 states.

TABLE I
MEAN AND VARIANCE OF THE ERROR CURVES IN FIGS. 9 AND 10

<table>
<thead>
<tr>
<th></th>
<th>25 points</th>
<th>49 points</th>
<th>97 points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>variance</td>
<td>mean</td>
</tr>
<tr>
<td>$K = 1$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>$K = 2$</td>
<td>1.032e-4</td>
<td>1.038e-9</td>
<td>1.037e-4</td>
</tr>
<tr>
<td>$K = 3$</td>
<td>4.201e-4</td>
<td>1.833e-8</td>
<td>4.109e-4</td>
</tr>
<tr>
<td>$K = 5$</td>
<td>1.715e-3</td>
<td>2.954e-7</td>
<td>1.620e-3</td>
</tr>
</tbody>
</table>

When comparing the proposed network to other self-organizing models, we have noted that the models by Grossberg [1] and Wang and Arbib [45] do not provide mechanisms to handle closed sequences with repeated points. Similarly, the ART-based models by Healy et al. [43] and Hagiwara [44] cannot deal with sequences with repeated/shared items. The self-organizing network by Kopecz [46] bears strong resemblance to our proposal in the sense that it uses the concept of spatial items linked via a time-delayed Hebbian rule to learn and recall temporal sequences. However, his neural model is unable to handle sequences with repeated items.

The superior performance of the proposed network, in comparison with the models described in the two previous paragraphs, is a result of the use of temporal context. Our approach involves the specification of a time-window of sufficient length $L$, say ten, and then training the network accordingly. Although, it is not a computationally efficient approach with respect to memory use, it has produced very good results. The automatic determination of the length $L$ is a major problem in models that adopt such a mechanism [5], [19], but recent works on temporal sequence processing have addressed this issue from the perspective of self-organizing algorithms (see, for example, [41]).

3) Choice of Network Parameters: The proposed network has a relatively high number of parameters (nine). They are, however, fairly easy to select and independent of each other in the sense that a change in one of them does not affect the others. If one always chooses $\sigma_{\text{max}} = \eta = \lambda = 1$, the number of parameters can be reduced to six. For their values, the following suggestions are given: 1) A value $\beta = 100$ or $1000$ always produces the exclusion behavior; 2) the simulations suggested $K = 1$ or $2$; 3) For these values of $K$, the value $d_{\text{min}} = 0.98$ results in acceptable accuracy in the case of faults; 4) the length $L$ of the local context is chosen on a trial-and-error basis, but it can be made adaptive (see, for example, [41]). The hint is to start with a small value, say two, and increase it if the network is unable to resolve possible ambiguities; 5) The number of output neurons, $m$, must guarantee the storage of all the $M$ original sequences and the associated $K - 1$ redundant sequences. If $M$ is known beforehand, $m = K \cdot \sum_{i=1}^{M} N^i$, where $N^i$ is the number of components of the sequence $I$. If $M$ is unknown, $m$ should be given initially a reasonably high value. If this number is eventually found to be insufficient, as in the case of continuous learning, a constructive technique should be used to add new neurons to the network [47]; and 6) a value $\epsilon = 10^{-5}$ for the similarity radius during training produces the desired results. For recall, this parameter can be set to $\epsilon = 0.1$ or 1 in order to handle noisy sequences.

V. Conclusion

Robot learning problems usually involve complex tasks characterized by: 1) a real-world system that integrates perception, decision-making and execution; 2) complex domains, yet the expense of using actual robotic hardware often prohibits the collection of large amounts of training data; and 3) real-time requirements in which decisions must be made within critical time constraints. These challenges to learning systems motivate considerably the search for suitable neural-network models for robot control.

In this paper a self-organizing network for temporal sequence learning and reproduction has been developed and applied to robotics. An essential feature of the model is that it copes directly with complex temporal sequences, processing them as easily as it does with simple ones. Handling such sequences, in fact, is indispensable for almost every kind of natural temporal behavior, in reading, writing, speech production, music generation, skilled motor behavior, and so on.

The network is very fast, accurate, and robust. A trajectory is stored after one single exposure of its states (one-shot learning) to the network. Feedforward weights learn the states of the trajectories, while lateral weights encode the transitions between successive trajectory states. For each state of the input trajectory, $K$ neurons are allocated to store it. The copies are not exactly
the same, the difference between them being determined by the proximity of their weight vectors to the current input state. Additional context units are used to resolve ambiguities during recall of the stored trajectories. These ambiguities arise when a trajectory has repeated states or shares states with others. The network is further evaluated on its ability to learn and recall noisy trajectories and its sensitivity to the trajectory sampling rate. There is a compromise between a higher value for the redundancy degree $K$ and the trajectory sampling rate. A higher $K$ is recommended for learning a trajectory with few states. A higher $K$ with high sampling rate result in poor performance of the network.

Despite being applied to manipulator control, the proposed model is general enough to be applied to other temporal sequence processing tasks. For instance, it has been previously used in processing sequences of letters in [38]. With minor changes, it can process stochastic sequences and can be applied to time-series prediction and robot navigation. In the former case, the network should model the underlying generating stochastic process by learning the temporal correlations of successive items via (9), and then use the learned representation to predict future values of the series. In the latter case, a robot can learn to associate a sequence of sensory experiences with certain actions as the robot randomly moves throughout the environment. An anti-Hebbian (negative $\lambda$) version of (9) could be used for blind source separation as in [27]. It is worth noting that in these applications the generated stochastic sequences are usually quite long. Thus, the one-shot learning approach should be replaced by a vector-quantization approach in order to obtain a compressed representation of the input sequence.

We further believe that the proposed neural-network model can serve both as a starting point for understanding some of the roles of temporal order in human motor behavior, and as an effective algorithm for robotics.

REFERENCES


Guilherme de A. Barreto (S’99) was born in Fortaleza, Ceará, Brazil, in 1973. He received the B.S. degree in electrical engineering from Federal University of Ceará in 1995, the M.S. degree in electrical engineering from the University of São Paulo in 1998 for the dissertation entitled “Unsupervised neural networks for temporal sequence processing,” and currently he is pursuing the D.Phil. degree in electrical engineering at the same university. The theme of his Ph.D. research is the use of self-organizing networks for identification and control of nonlinear dynamical systems.

His research interests include theory and applications in the areas of neural networks, temporal sequence processing, robotics, time series analysis, and nonlinear dynamic systems.

Aluizio F. R. Araújo was born in Recife, Pernambuco, Brazil, in 1957. He received the B.S. degree in electrical engineering from Federal University of Pernambuco in 1980, the M.S. degree in electrical engineering from the State University of Campinas, Campinas, Brazil, in 1988, and the D.Phil. degree in computer science and artificial intelligence from the University of Sussex, Sussex, U.K., in 1994.

He worked in São Francisco Hidroelectrical Company for five years and in 1998 he became an Assistant Professor at University of São Paulo, São Paulo, Brazil, where in 1994 he was promoted to Adjunct Professor. His research interests are in the areas of neural networks, machine learning, robotics, dynamic systems theory, and cognitive science. He has published more than 55 papers in national and international journals and conferences.