52 — 10/01/01

Proceedings of the

Second ICSC Symposium on

ENGINEERING OF INTELLIGENT SYSTEMS

EIS 2000

June 27 - 30, 2000
University of Paisley, Scotland, U.K.

Editor: C. Fyfe

ISBN 3-906454-21-5

ICSC
International Computer Science Conventions
Canada / Switzerland

A Self-Organizing Neural Network for Learning and Recall of Complex Robot Trajectories

Aluizio F. R. Araújo Guilherme de A. Barreto

Universidade de São Paulo
Departamento de Engenharia Elétrica
C. P. 359, 13560-970, São Carlos, SP, BRASIL
{gbarreto, aluizioa}@sel.eesc.sc.usp.br

Abstract. We propose an unsupervised neural network model to learn and recall complex robot trajectories. Two cases are considered: (1) A single trajectory in which a particular arm configuration may occur more than once, and (2) trajectories sharing states with other ones – they are said to contain a shared state. Hence, ambiguities occur in both cases during recall of such trajectories. The proposed model consists of two groups of synaptic weights trained by competitive and Hebbian learning laws. They are responsible for encoding spatial and temporal features of the input sequences, respectively. Three mechanisms allow the network to deal with repeated or shared states: local and global *context units, neurons disabled to learn, and* redundancy. The network produces the current and the next state of the learned sequences and is able to solve ambiguities. The model is simulated over various sets of robot trajectories in order to evaluate learning and recall, trajectory sampling effects and robustness.

Keywords: Unsupervised learning, neural networks, temporal sequences, context, robotics.

1. Introduction

A common problem in robotics is *trajectory tracking*, in which a robot is required to follow accurately a continuous pathway (Craig, 1989). Such a task is mainly pre-programmed such that the arm positions are stored in the controller memory for later recall. This method may become time consuming and uneconomical for complex trajectories (Chen et al., 1996).

The research in artificial neural network (ANN) models makes it possible to investigate solutions for complex problems in robotics following different learning paradigms (Zomaya & Nabhan, 1994). Tracking can be handled within the framework of artificial neural networks for temporal processing since trajectories can be seen as spatiotemporal sequences of arm positions.

The unsupervised learning paradigm has appealing properties for its use in robotics and temporal sequence processing. The behavior in unsupervised neural networks emerges by means of a self-organizing process, which reduces substantially the robot programming burden that accounts even for one third of the total cost of an industrial robot system (Heikkonen & Koikkalainen, 1997).

An important issue, usually not addressed in simulations and tests reported by the neural network literature, is the learning of multiple robot trajectories (Chen et al., 1996). 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 to learn and retrieve multiple trajectories. The various unsupervised neural models to robot control may be divided into three main approaches: (i) learning of *perception-action* trajectories, (ii) learning of *robot state* trajectories, and (iii) planning of *robot state* trajectories.

The first approach, also called sensory-motor learning, associates sensory data with desired actions (Bugmann et al. 1998). It is used when a mobile robot is required to explore the world to build a model for it. As the robot navigates, it experiences a long sequence of perception-action pairs. This approach is not stable against deviations of the trajectory. If the robot finds itself in an untrained position, off the trajectory, no adequate control action may be produced (Bugmann *et al*, 1998). Examples of this approach are the models proposed by Denham & McCabe (1995), Rao & Fuentes (1996), Owen & Nehmzow (1996) and Heikkonen & Koikkalainen (1997).

In the second approach, the network must associate consecutive states of a trajectory and store these transitions for total or partial reproduction of the memorized trajectory. For purpose of recall, the network receives the current state of the robot and responds with the next one. See for example the models by Hyötyniemi (1990), Althöfer and Bugmann (1995), Bugmann *et al* (1998) and Barreto & Araújo (1999a, b).

The third approach entails the creation of a robot trajectory given only its initial and final positions. The robot receives sensory information from the workspace and constructs an inverse kinematic mapping. This approach is used in Kuperstein & Rubinstein (1989), Martinez *et al* (1990), and Walter & Schulten (1993).

In this paper we aim at emphasizing 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 robot trajectories. The contribution of this work is twofold: (i) development of Hebbian learning rules to process spatiotemporal patterns and (ii) application of such a model to control robots involved in complex tracking tasks. The learning algorithm is evaluated through simulations on complex trajectories.

The paper is organized as follows. In Section 2, we present the model. In Section 3, we evaluate the performance of the model through computer simulations and discuss the main results. We conclude the paper in Section 4.

2. The Proposed Neural Model Description

The architecture of the model is shown in Figure 1. It is a two-layer network composed of an input and an output layer which is responsible for the whole processing. The model has feedforward and feedback connections that play different roles in its dynamics.

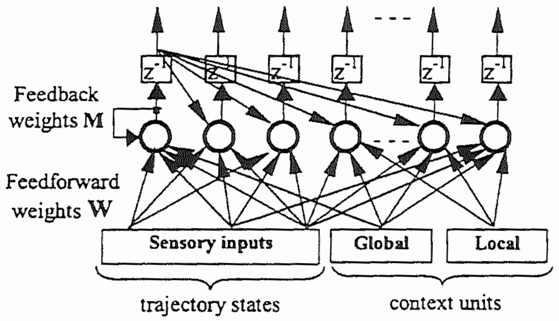


Figure 1. **The architecture of the proposed model.**

The *input* comprises sensory and context units. The first set of units receives the trajectory state at time step t and propagates it towards the output layer. By trajectory state at time t we mean end-effector position, joint angles and applied joint torques. For example, for a six degree of freedom robotic arm moving in 3D space, we need three units for the end-effector position (x, y, z), six units for the joint angles θ_i, i=1, ..., 6, and six units for the joint applied torques τ_i, i=1, ..., 6. Two types of trajectories are considered: (i) closed (initial state = final state), and (ii) open (initial state $\neq$ final state). Pre-processing of the input data is not required.

Context units are necessary when a single trajectory has repeated patterns, or multiple sequences have shared states. Global context (*time-invariant*) is always set to the final spatial position of a given trajectory. Usually, this information is supplied as task specification. For open trajectories, fixed context units are sufficient for correct storage and recall (Barreto & Araújo, 1999a, b). However, for closed paths with crossing points, additional information is necessary. Then, local context (*time-variant*) is added to the network input. These units are always set to the spatial coordinates of the states that precedes the one just delivered to the network. The inclusion of time-varying context units in this work allows the network to encode both closed and open trajectories, increasing the model applicability.

The *output neurons* represent the current and the next states in a particular sequence. The weight vectors associated with the most activated neurons are then used as control signals to place the robot arm at the desired configuration.

The *synaptic weights* consist of feedforward (or interlayer) weights and feedback (or intralayer) weights. The interlayer and intralayer weights are updated by competitive and Hebbian learning rules respectively. The feedforward weights connect each input unit to each output neuron. They encode the spatial configuration of the robot arm at a specific time step. Feedforward weights are initialized randomly with numbers between 0 and 1. The intralayer coupling structure encodes the temporal order of the patterns in a sequence. The Feedback weights are initialized with zeros, indicating no temporal associations at all.

The two groups of synaptic weights are updated during a single pass of an entire trajectory in which each state is read once. This means that a sequence with N_c components requires N_c training steps. An input state is compared with each feedforward weight vector through Euclidean distance. The group of weight vectors closest to the input vector is selected to be updated. Mathematically, we have:

$$\begin{aligned} v_1(t) &= \arg\min_j \left\{ f_j(t) \left\| x_i - w_{ji} \right\| \right\} \quad \text{for all } j \\ v_2(t) &= \arg\min_j \left\{ f_j(t) \left\| x_i - w_{ji} \right\| \right\} \quad \forall j \notin \{v_1\} \qquad (1) \\ &\vdots \qquad\qquad\qquad \vdots \qquad\qquad \vdots \end{aligned}$$

$$v_K(t) = \arg\min_j \{f_j(t) \| x_i - w_{ji} \| \} \quad \forall j \notin \{v_1, \ldots, v_{K-1}\}$$

where $v_1, \ldots, v_K$ are indices ranking the proximity between the weight vectors and the current input. Thus, for each input vector, a cluster of K neurons is chosen to encode it, similarly to neighboring neurons in Kohonen's SOM (1997). However, unlike the neurons in SOM, the output neurons do need to be in physical neighborhood. Redundancy results in a network more resilient to noise and tolerant to neuronal faults. For recall, we always set $K = 1$.

The function $f_j(t)$ is a penalty factor defined as:

$$f_j(t) = \begin{cases} \alpha, & \text{if } j \in \{v_1, \ldots, v_K\} \\ f_j(t), & \text{otherwise} \end{cases} \quad (2)$$

where $f_j(0) = 1$ and $\alpha >> 1$. This function is used to exclude a winning neuron from subsequent competitions to guarantee that each point of the trajectory is encoded by different neurons.

The neuronal activities $a_j(t)$ and outputs $y_j(t)$ are determined as follows:

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

where $0 < \gamma < 1$ and $A \geq 1$. According to Eq. (3), the K winners at time step t receive an amount of activity proportional to its rank as determined in Eq. (1). Note that due to Eq. (2), $a_j(t)^T a_r(t-1) = 0$, for $i \neq r$. The output $y_j(t)$ is defined as follows:

$$y_j(t) = g\left(\sum_{r=1}^{N} m_{jr}(t) a_r(t) \right) \quad (4)$$

where $g(u) \geq 0$ so that $dg(u)/dt > 0$, $m_{jr}(t)$ is the intralayer connection weight between the output neurons r and j, and N is the number of output neurons. The highest output value determines the weight vector to be sent to the robot controller

Following 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:

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

where $\delta \approx 1$ is the learning rate. Note that units with activities $a_j(t)$ equal to zero do not learn (see Eq. (2)).

The successive winners are linked in the correct temporal order through a lateral coupling structure. These feedback weights are updated according to the following learning rule (Barreto & Araújo, 1999b):

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

where λ is the feedback learning rate. The activation pattern of the previous competition, $a_r(t-1)$, is made available through time delays. Equation (6) is a Hebbian learning rule (Hebb, 1949) that creates temporal associations between consecutive states of the input trajectory. Time in Hebbian learning rules plays an essential role in psychology (Montague & Sejnowski, 1994), biology (Wallis, 1998), route learning and navigation (Schölkopf & Mallot, 1995) and blind source separation (Girolami & Fyfe, 1996).

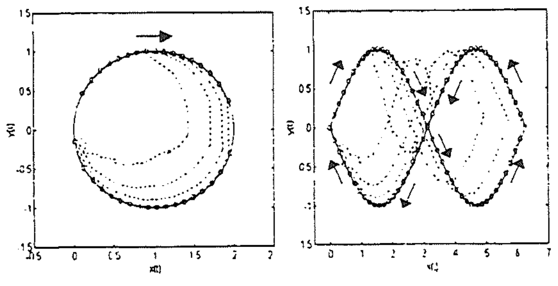
It is important to emphasize that two elements are essential for sequence recall. First, a mechanism of short-term memory (Mozer, 1993) to enable extraction and storage transitions from one pattern to its successors in the sequence (see Eq. 6). Second, the activation dynamics must be defined to mimic the previously learned sequence by moving through correct sequence of stored states (see Eqs. (3) and (4)). This way, the next state of a stored sequence is recalled every time an input vector matches one of the stored patterns. The weight vector of this "next state" neuron supplies the robot with the next spatial position, the associated joint angles, and the applied torques. Once a robot has reached its next position, new sensor readings are fed back to the neural network input that generates the next state of the sequence. This process continues until the entire trajectory is reproduced.

3. Simulations

We evaluate the proposed neural network in storing and recalling different types of trajectories. We consider closed 2D robot trajectories and open 3D robot trajectories of a PUMA 560 with 6 DOF generated by the toolbox *ROBOTICS* of Matlab (Corke, 1996). The circular paths have 20, 35, 70 and 100 points. The figure-eight trajectories the sequences are 20, 40, 80 and 100 points long. Each open trajectory has 11 states including the initial and the final ones. These trajectories are presented sequentially, one after the other. The network performance is evaluated in tracking tasks by means of the root mean square error (RMSE).

3.1. The Learning of Closed Trajectories

Choice of Learning Rate δ. This simulation intends to show how the learning rate influences the storage accuracy of the proposed model. We have trained the network on the circular and figure-eight trajectories for 4 different values of the learning parameter $\delta = 0.45$, 0.75, 0.90 and 0.99. The other parameters were set to $K = 1$, $\alpha = 1000$, $\lambda = 0.8$, $\gamma = 0.99$, and $N = 100$. The number of sensory inputs is 6, because we need 2 units for the end-effector position, 2 for the joint angles and 2 for the applied joint torques. In addition, we need 2 more units for the local context and 2 more for the global context. The feedforward weights were randomly initialized between 0 and 1 and the feedback units are initialized with zeros. The same initial weights are used for all values of δ. The results are plotted in Figure 2a and 2b for a circular trajectory (35 states) and a figure-eight trajectory (80 points), respectively.

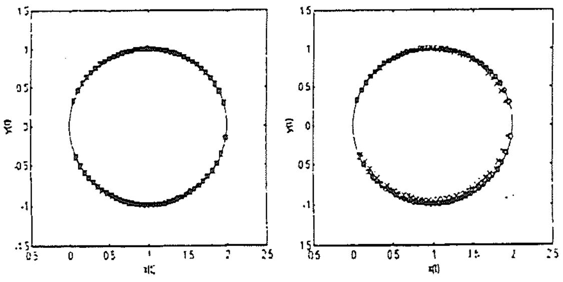


(a) (b)

Figure 2. Accuracy in learning closed trajectories for δ = 0.45, 0.75, 0.90 and 0.99. Inner trajectories has lower values for δ. Arrows indicate the direction of movement.

The errors for the circular trajectories were 2.1731 (δ=0.45), 0.9655 (δ=0.75), 0.3776 (δ=0.90) and 0.0388 (δ=0.99). The errors for the figure-eight trajectory were 12.0304 (δ=0.45), 5.4090 (δ=0.75), 2.1716 (δ=0.90) and 0.2185 (δ=0.99). As expected, the RMSE decrease as δ approaches 1. Thus, if accuracy is required, δ must be nearly or equal to 1. It is an important achievement since the robot controller must be supplied with precise signals from the network.

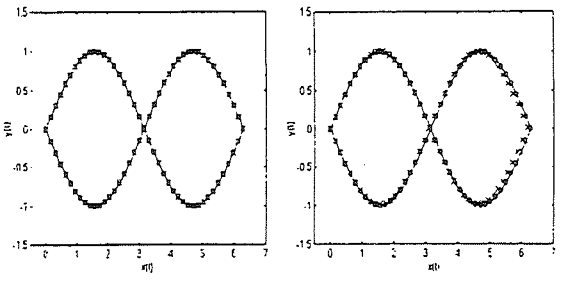
Influence of Redundancy on Fault-Tolerance. In this simulation, we show how a trajectory is stored by the K first winning neurons, and why such redundancy is useful in cases of neuronal fault. We chose $K = 3$, i.e., each trajectory state is encoded by 3 different neurons. The other parameters were set to $\delta = 1$, $\alpha = 10^6$, $\lambda = 0.8$, $A = 1$, $\gamma = 0.99$, $N = 525$. The results for a circular trajectory with 70 points are shown in Figure 3. Figure 3a illustrates the input (circles) and the stored/retrieved trajectory (crosses) encoded by the first winner neuron, while Figure 3b presents the result for the third winner unit. The RMSE values for the stored trajectories are 0.00 (1st winner), 0.1438 (2nd winner), and 0.2877 (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 = 0.1438. In the extreme and unlike case of total collapse of the first and second winners, the third winners would be used by the network, yielding RMSE = 0.2877.



(a) (b)

Figure 3. Redundancy effects in learning circular trajectories for the: (a) 1st winner and (b) 3rd winner.

An example for a figure-eight trajectory with 80 points is plotted in Figure 4. This sequence has a crossing position at coordinates (3.13, 0.00). To correctly recall the trajectory, the context units are set to the coordinate of the state which immediately precedes the current sensory input. The RMSE values are: 0.00 (1st winner), 0.2233 (2nd winner), and 0.4452 (3rd winner).



(a) (b)

Figure 4. Redundancy effects in learning figure-eight trajectories for the: (a) 1st winner and (b) 3rd winner.

We conclude that, for tracking purpose, the robot controller must use the trajectories in Figure 3a and 4a. In the case of neuronal 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 neuronal failures, by simply adopting $\delta = \gamma = A = 1$. However, this would turn the network much like a fault-tolerant conventional storage-and-recall device without the ability to generalize.

Influence of Sampling Rate and Redundancy on Noise-Tolerance. The previous simulations considered noise-free trajectories. However, tolerance to noise is a desirable property for any controller of a real robot. The tolerance to noise of our network is tested by adding different amounts of zero mean Gaussian white noise to the trajectory states and calculating the RMSE value. The noise variance ranges from 0.001 to 0.1.

Another issue to be addressed is the sampling rate effects on the network performance. In this case, we evaluate how the network responds to noisy trajectory as one varies the redundancy degree and the number of points of the input trajectory. We simulated the network for K = 3, 4 and 5. Figures 5 show typical results for circular trajectories. Note that lower values for RMSE (solid) are obtained by choosing K = 5. The worst results were obtained for K = 3 (dashed-dotted) and 4 (dotted). Results for figure-eight trajectories (not shown) follow the same pattern.

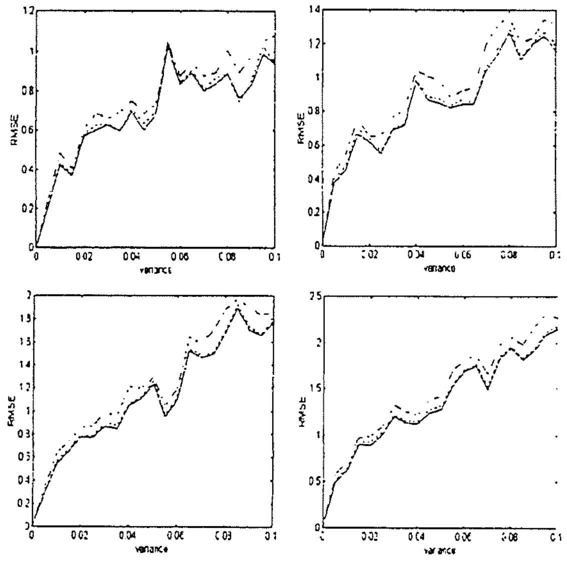


Figure 5: Noise Tolerance of the network trained on circular trajectories for different sampling rates: (a) 20 points, (b) 35 points, (c) 70 points and (d) 100 points.

Note that, at some limit, higher values of K do not imply in lower values of RMSE. On the contrary, it can result in higher values of RMSE, especially when the number of points in a sequence increases. This is explained by noting that as the distance between consecutive points decreases due higher sampling rates, the chance of the network to choose an incorrect winner due to noise also increases.

3.2. The Learning of Many Robot Trajectories

In order to test the algorithm ability to encode multiple open trajectories, the following considerations are made: (1) the final state of a given trajectory are known and (2) any trajectory must contain at least one crossing state with all the other ones at any intermediate state.

The parameters were set to α = 1000, γ = 0.95, δ = 1.0, λ= 0.8, N = 70 and three trajectories were trained sequentially. Figure 6 shows the results following the training stage on two trajectories which have a crossing point at (0.20, 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 Figure 6a is 0.0024. This illustrates the ability of Eq. (5) in accurately encoding an input pattern in only one iteration. The letters I and F indicates the initial and final points of the trajectory, respectively.

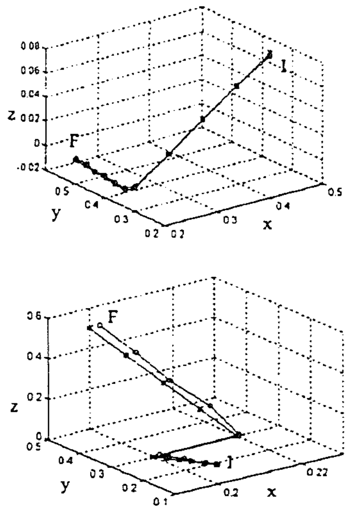


Figure 6. Two trained trajectories with one shared point. Desired trajectory (circles), retrieved trajectory (crosses).

Figure 7 shows the trajectories in Figure 6 in a simulator we have developed in order to have an idea of the trajectory behavior in robot workspace.

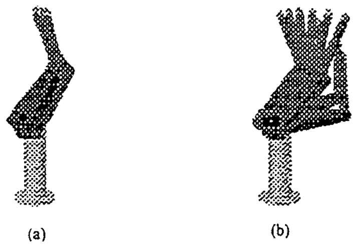


Figure 7. Trajectories of Figure 6 in a simulated robot workspace.

Figures 8 and 9 show the shoulder, elbow and wrist joint angles and torques associated with each state of the stored trajectories. Similarly, the algorithm was able to encode them with a small error. The desired and stored values are practically the same. Note that the algorithm can learn the input independently of its magnitude and sign. Also note that the network responds equally nice to trajectories with smooth curvature (circular and figure-eight types) as well as to those with abrupt changes of direction (Figure 6 and 7).

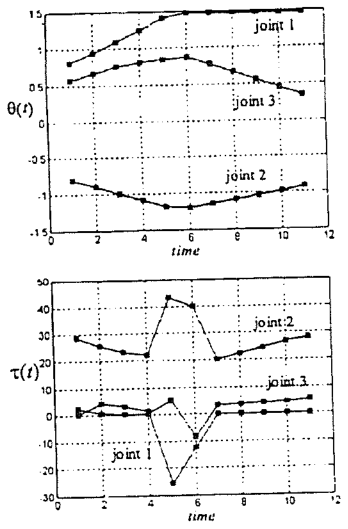


Figure 8. The joint angles and torques for the trajectory shown in Figure 6a. Angles (rad) and torques (N.m).

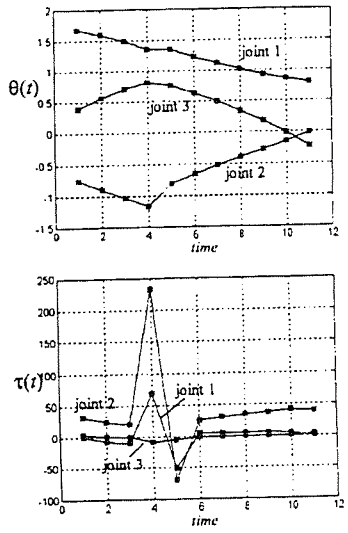


Figure 9. The joint angles (a) and (b), and torques (c) and (d) associated with trajectory in Figure 6b.

Figure 10 illustrates the fault-tolerance for this type of trajectories. In this case, we simulated neuronal faults as we did for circular and figure-eight trajectories, i.e., by excluding all the first winning neurons of each one of the three trajectories. The second winners are now liable for the retrieval of the stored sequence. Even so, the network was able to reproduce the trajectories correctly. This result justifies the use of more than one neuron during the learning of the feedforward weights.

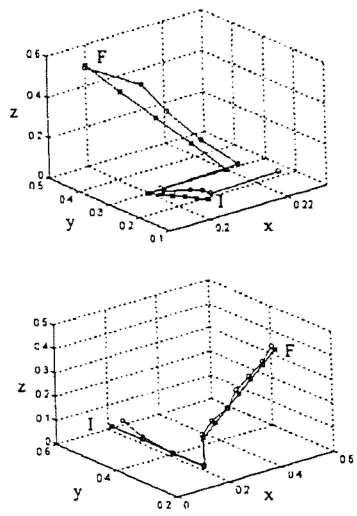


Figure 10. The same trained trajectories of Figure 10. However, in this case, a fault in one of the neurons encoding each trajectory state is simulated.

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 equally learned with small tracking error.

4. Conclusion and Further Work

An unsupervised model to learn and recall robot trajectories is proposed. This simple neural network accurately stores and retrieves open and closed complex robot trajectories. Compared to the approaches by Althöfer & Bugmann (1995), Bugmann *et al* (1998) our model advances in the following points: (i) It is noise and fault tolerant, (ii) the state transitions are not pre-wired but rather learned through Hebbian learning, (iii) it is able to learn new trajectories incrementally, and (iv) it is able to handle open and closed trajectories sampled at different rates. The proposed model can be viewed as an extension of our previous work (Barreto & Araújo, 1999a, b). The current one incorporates the ability to learn and recall closed trajectories with or without crossing points.

It is worth emphasizing that the proposed model, unlike other common neural network approaches to temporal sequence processing, does not require two networks to learn the invariant temporal order of the input sequence; that is, two networks learning exactly the same spatial patterns which are usually linked in an *ad hoc* (non-adaptive) manner. In our model, the spatial patterns are stored in a feedforward layer of weights and linked in time through lateral connections in a self-organizing fashion.

At the moment we are pursuing mechanisms to better use memory resources, since, due to Eq. (2), for every state occurring more than once, K similar copies of it are stored by K different neurons. Also, the implementation of the proposed model in a real robot is being evaluated.

Acknowledgements

The authors thanks FAPESP (project Nr. 98/12699-7) for financial support.

References

Althöfer, K. & Bugmann, G. (1995). Planning and learning goal-directed sequences of robot arm movements. *Proc. of the Int. Conf. on Artificial Neural Networks* (ICANN'95), Paris, France, vol. 1, pp. 449-454.

Barreto, G. A. & Araújo, A. F. R. (1999a). Fast learning of robot trajectories via unsupervised neural networks. *Proc. 14th IFAC World Congress*, Beijing, China, pp. 373-378, Pergamon Press, Oxford.

Barreto, G.A. & Araújo, A.F.R. (1999b). Unsupervised learning and recall of temporal sequences: An application to robotics. *Int. Journal of Neural Systems*, 9(3):235-242.

Bugmann, G., Koay, K. L., Barlow, N., Phillips, M. & Rodney, D. (1998). Stable encoding of robot trajectories using normalized radial basis functions: Application to an autonomous wheelchair. *Proc. of the 29th Int. Symp. on Robotics* (*ISR*'98), Birmingham, UK, pp. 232-235.

Chen, C.Y, Mills, J.K. & Smith, K.C. (1996). Performance improvement of robot continuous-path operation through iterative learning using neural networks. *Machine Learning*, 23:75-105.

Corke, I. (1996). A Robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24-32.

Craig, J.J. (1989). *Introduction to Robotics: Mechanics and Control*, 2nd edition, Reading, MA: Addison-Wesley.

Denham, M. J. & McCabe, S. L. (1995). Robot control using temporal sequence learning. *Proc. of the World Congress on Neural Networks*, vol. 2, 346-348.

Girolami, M. & Fyfe, C. (1996). A temporal model of linear anti-Hebbian learning. *Neural Proc. Letters*, 4:139-148.

Hebb, D.O. (1949). *The organization of behavior*. Wiley.

Heikkonen, J. & Koikkalainen, (1997). Self-organization and autonomous robots. In: *Neural Systems for Robotics*, O. Omidvar and van der Smagt (Eds.), Academic Press, 297-337.

Hyötyniemi, H. (1990). Locally controlled optimization of spray painting robot trajectories. *Proc. IEEE Int. Workshop on Intelligent Motion Control*, Istanbul, Turkey, pp. 283-287.

Kohonen, T. (1997). *Self-organizing maps*. 2nd edition, Springer-Verlag.

Kuperstein, M. & Rubinstein, J. (1989). Implementation of an adaptive neural controller for sensory-motor coordination. *IEEE Control Systems Magazine*, 9(3):25-30.

Martinetz, T.M., Ritter, H.J. & Schulten, K.J. (1990). Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Trans. Neural Networks*, 1(1):131-136.

Montague, R. & Sejnowski, T. J. (1994). The predictive brain: temporal coincidence and temporal order in synaptic learning mechanisms. *Learning & Memory*, 1:1-33.

Mozer, M. C. (1993). Neural net architectures for temporal sequence processing. In: *Predicting the Future and Understanding the Past*, A. Weigend & N. Gershenfeld (Eds.), Redwood City, CA: Addison-Wesley, 243-264.

Owen, C. & Nehmzow, U. (1996). Route learning in mobile robot through self-organization. *Proc. of Eurobot. 96*, IEEE Computer Society Press.

Rao, R.N. & Fuentes, O. (1996). Learning navigational behaviors using a predictive sparse distributed memory. *Proc. of From Animals to Animats: the 4th Int. Conference on Simulation of Adaptive Behavior*, pp. 382-390, MIT Press.

Schölkopf & Mallot (1995). View-based cognitive mapping and path planning. *Adaptive Behavior*, 3:311-348.

Wallis, G. (1998). Temporal order in human object recognition. *J. Biol. Syst.* 6(3):299-313.

Walter, J.A. & Schulten, K.J. (1993). Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Trans. on Neural Networks*, 4(1):86-95.

Zomaya, A.Y. & Nabhan, T.M. (1994). Trends in neuroadaptive control for robot manipulators, In: *Handbook of Design, Manufacturing and Automation*, R.C. Dorf & A. Kusiak (Eds.), pp. 889-917, Wiley & Sons: New York, U.S.A.