Predictive Modeling and Planning of Robot Trajectories Using the Self-Organizing Map

Guilherme A. Barreto[1] and Aluízio F.R. Araújo[2]

[1] Universidade Federal do Ceará (UFC)
Departamento de Engenharia de Teleinformática
Campus do Pici, Centro de Tecnologia, Fortaleza, CE, Brazil
guilherme@deti.ufc.br
http://www.deti.ufc.br/~guilherme
[2] Universidade Federal de Pernambuco (UFPE)
Centro de Informática - CIn
Departamento de Sistemas da Computação
Av. Professor Luís Freire, s/n, Cidade Universitária
50740-540, Recife, PE, Brazil
Recife/PE, Brazil
aluizioa@cin.ufpe.br

Abstract. In this paper, we propose an unsupervised neural network for prediction and planning of complex robot trajectories. A general approach is developed which allows Kohonen's Self-Organizing Map (SOM) to approximate nonlinear input-output dynamical mappings for trajectory reproduction purposes. Tests are performed on a real PUMA 560 robot aiming to assess the computational characteristics of the method as well as its robustness to noise and parametric changes. The results show that the current approach outperforms previous attempts to predictive modeling of robot trajectories through unsupervised neural networks.

1 Introduction

Prediction of robot trajectories is a recent research topic in the Neural Network literature [1], [2], [5], [6], [7]. In this context, prediction is understood as the capacity of an artificial neural network (ANN) to determine the next state (configuration) of robot with respect to a particular trajectory. The training process of this ANN makes it able to execute complex robotic control tasks, such as autonomous trajectory planning, avoidance of obstacles in an environment, and nonlinear predictive control.

The early unsupervised neural approaches to learning robotic trajectories employed *static* ANNs [12], [13], i.e., models that are understood as systems without memory. In this situation, the neural algorithm has to learn a mapping considering input-output patterns that occur at the same instant of time, such as forward and inverse kinematics. However, this approach has limited applicability to robotic tasks which require the learning of both spatial and temporal relationships, such as trajectory planning. Static

R. Orchard et al. (Eds.): IEA/AIE 2004, LNAI 3029, pp. 1156-1165, 2004.

ANNs can also be used to trajectory planning, but in this case, the temporal order of the trajectory states is not learned by the network itself, but rather predefined by the robot operator.

This paper presents a neural network model based on the Self-Organizing Map (SOM) algorithm [10] to predictive modeling of complex robot trajectories. In this approach, the neural network automatically learns the temporal order of the trajectory states through associative memory mechanisms. The proposed model is then used to plan and control trajectories of a real 6-DOF PUMA 560. Moreover, the tests were also designed to estimate appropriate parameters and to evaluate the model's robustness to input noise and parametric changes.

The remainder of this paper is organized as follows. A brief summary on the SOM is presented in Section 2, in which we show how it is used to build static input-output mappings. In Section 3, a temporal associative schema, which extends the SOM algorithm to learn dynamical mappings, is described. Such a strategy is employed to plan and control trajectories of the PUMA 560 and the implementation of this method is discussed in Section 4. Section 5 concludes the paper.

2 Self-Organizing Maps and Dynamic Mappings

The SOM is an unsupervised neural network algorithm developed to learn neighborhood (spatial) relationships of a set of input data vectors. Each neuron i has a weight vector $\mathbf{w}_i \in \Re^n$ with the same dimension of the input vector $\mathbf{x} \in \Re^n$, and all neurons are usually arranged in a two-dimensional array, called output layer. The SOM learning algorithm can be summarized in two basic steps:

1. Find the winning neuron at time t: $i^*(t) = \arg\min_{\forall i} \left\| \mathbf{x}(t) - \mathbf{w}_i(t) \right\|$ (1)

2. Then, update the weight vectors: $\Delta \mathrm{w}_i(t) = \eta(t) h(i^*, i; t)[\mathrm{x(t)} - \mathrm{w}_i(t)]$ (2)

where $0 < \eta(t) < 1$ is the learning rate and $h(i^*, i; t) = \exp(-\| \mathbf{r}_{i^*}(t) - \mathbf{r}_i \|^2 / 2\sigma^2(t))$ is a Gaussian neighborhood function in which $\mathbf{r}_i(t)$ and $\mathbf{r}_{i^*}(t)$ denote the positions of the neurons i and i^* in the output array. For the sake of convergence, the parameters $\eta(t)$ and $\sigma(t)$ should decrease in time, for example, in a linear basis: $\eta(t) = \eta_0(1\text{-}t/T)$ and $\sigma(t) = \sigma_0(1\text{-}t/T)$, where η_0 and σ_0 are the initial values of $\eta(t)$ and $\sigma(t)$, respectively. T denotes the maximum number of training iterations. Another common choice for the neighborhood function is the rectangular one (also called bubble): $h(i^*, i; t) = 1$, if $i \in V_{i^*}(t)$, and $h(i^*, i; t) = 0$, otherwise. The set $V_{i^*}(t)$ contains all neurons in the neighborhood of the winning neuron i^* at time t. As in the Gaussian case, the size of $V_{i^*}(t)$ should also decay in time for convergence purposes.

The work by [13] extended the SOM network to learn mappings from input-output pairs of static patterns. Such static mappings are often described by:

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{u}(t)) \tag{3}$$

where $\mathbf{u}(t) \in \Re^n$ denotes he input vector and $\mathbf{y}(t) \in \Re^m$ is the output vector. More recently, the work by [4] generalized the model of Walter and Ritter in order to encode dynamic mappings, such as those described in [9]:

$$\mathbf{y}(t+1) = \mathbf{f}[(\mathbf{y}(t), \ldots, \mathbf{y}(t-n_y+1); \mathbf{u}(t), \ldots, \mathbf{u}(t-n_u+1)] \tag{4}$$

where n_u and n_y are the orders of the input and output memories. Equation (4) indicates that the output at time instant t+1 depends on the n_y past outputs and n_u past inputs. Usually, the mapping **f**(.) is nonlinear and unknown.

In order to establish temporal associations between consecutive patterns in a temporal sequence, neural networks need to retain information about past sequence items [3]. Such a retention mechanism, called *short-term memory* (STM), encodes temporal order and/or temporal dependencies between successive sequence patterns. STM can be implemented by different strategies, the simplest being the so-called *tapped delay line*, understood as a sliding "time window" over the input sequence within which a number of successive samples are concatenated into a single pattern vector of higher dimensionality. In the following, we use delay lines as the STM mechanism to allow the SOM to learn input-output dynamical mappings.

3 Temporal Associative Memory

In order to approximate **f**(.), the input and output vectors of a time series {**u**(t), **y**(t)}, t = 1, …, N are organized into a single input vector to presented to the SOM. The first component of $\mathbf{x}^{in}(t)$ represents the actual input information of the mapping whereas the second piece, $\mathbf{x}^{out}(t)$, corresponds to the desired output information of the same mapping. The input and weights vectors are then redefined as

$$\mathbf{x}(t) = \begin{pmatrix} \mathbf{x}^{in}(t) \\ \mathbf{x}^{out}(t) \end{pmatrix} \qquad \text{and} \qquad \mathbf{w}_i(t) = \begin{pmatrix} \mathbf{w}_i^{in}(t) \\ \mathbf{w}_i^{out}(t) \end{pmatrix} \tag{5}$$

During the training phase, the winning neurons are found based solely on the input component of the extended input vector **x**(t):

$$i^*(t) = \arg\min_{\forall i} \left\| \mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t) \right\| \tag{6}$$

However, the weight updates consider both parts of the input vector:

$$\Delta \mathbf{w}_i^{in}(t) = \eta(t) h(i^*, i; t)[\mathbf{x}_i^{in}(t) - \mathbf{w}_i^{in}(t)] \tag{7}$$

$$\Delta \mathbf{w}_i^{out}(t) = \eta(t) h(i^*, i; t)[\mathbf{x}_i^{out}(t) - \mathbf{w}_i^{out}(t)] \tag{8}$$

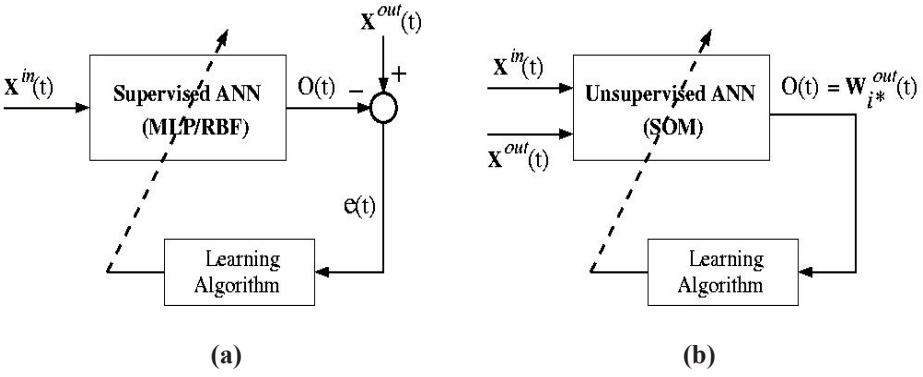


Fig. 1. Difference between (a) supervised and (b) unsupervised (MATQV) learning of input-output mappings.

Thus, by means of Equations (5)-(8) the SOM model learns to associate the input signals with the output ones, while simultaneously performing vector quantization of the input and output spaces. Bearing that in mind, this technique is called *Vector Quantized Temporal Associative Memory* (VQTAM). The VQTAM may be used for approximating different types of mapping depending on the nature of the vectors $\mathbf{x}^{in}(t)$ and $\mathbf{x}^{out}(t)$. For example, the VQTAM formulation of the system defined in Equation (4) is given by

$$\mathbf{x}^{in}(t) = [\mathbf{y}(t), \ldots, \mathbf{y}(t - n_y + 1); \mathbf{u}(t), \ldots, \mathbf{u}(t - n_u + 1)]^T \quad \text{and} \quad \mathbf{x}^{out}(t) = \mathbf{y}(t+1) \tag{9}$$

Once the mapping is learned, the VQTAM may be used to estimate the output values of this mapping by the equation stated below

$$\hat{\mathbf{y}}(t+1) = \mathbf{w}_{i^*}^{out}(t) \tag{10}$$

where the winning neuron $i^*(t)$ is determined as in Equation (6). The prediction process is repeated M times until a time series of estimated values is available.

Training the VQTAM is characterized by simultaneous presentation of the vectors $\mathbf{x}^{in}(t)$ and $\mathbf{x}^{out}(t)$ and the absence of an explicit computation of an error signal (Fig. 1b). On the other hand, a supervised learning algorithm, such as those used in RBF and MLP networks, employ an error to guide the training, so that only the vector $\mathbf{x}^{in}(t)$ is used as input of the network and the vector $\mathbf{x}^{out}(t)$ is the desired output (Fig. 1a) needed to compute the error.

3.1 Predictive Modeling of Robot Trajectories

The VQTAM makes it possible to learn and to reproduce complex mappings, such as those responsible for the generation of robot trajectories. In particular, the focus of this study is the prediction of joint angles associated with a given robot trajectory of the 6-degree-of-freedom PUMA 560 robot, and their posterior reproduction for trajectory planning purposes. Let the vector $\boldsymbol{\theta}(t) = [\theta_1(t) \ldots \theta_6(t)]^T$ represent the vector of joint angles at time t. In this context, Equation (4) reduces to

$$\boldsymbol{\theta}(t+1) = \mathbf{f}[(\boldsymbol{\theta}(t), \ldots, \boldsymbol{\theta}(t - n_\theta + 1)] \tag{11}$$

where n_θ is the model order, also called memory parameter. This formulation allows the SOM to learn the kinematics of the manipulator while simultaneously encoding the temporal order of the trajectory states. In this case, the vectors $\mathbf{x}^{in}(t)$ and $\mathbf{x}^{out}(t)$ are the following:

$$\mathbf{x}^{in}(t) = [\boldsymbol{\theta}(t), \ldots, \boldsymbol{\theta}(t - n_\theta + 1)]^T \quad \text{and} \quad \mathbf{x}^{out}(t) = \boldsymbol{\theta}(t+1) \tag{12}$$

where the determination of the winning neuron follows Equation (9) and the weight adjustments are determined by Equations (10) and (11). The estimate for the next vector of joint angles is then given by

$$\hat{\boldsymbol{\theta}}(t+1) = \mathbf{w}_{i^*}^{out}(t) \tag{13}$$

Such an estimate can then used as setpoint for autonomous planning and control of the manipulator (Fig. 2). In the experiment to be described in the next section, the robot operator defines the initial state of the trajectory for $t = 0$, and the neural network generates the next state. Such a state is sent to the robot that moves itself to the desired position. The new configuration of the joints of the PUMA is then measured and fed back to the network that produces the next state. This procedure is repeated until the end of the trajectory is reached. Due to the inherent perturbations, very often the robot moves to a neighborhood of the target position, thus the neural networks has to be robust to these perturbations. That is, the generated responses should be stable, i.e., close enough to the states of the learned trajectory.

It is also worth emphasizing the differences between the predictive approach to autonomous trajectory planning and the conventional *look-up table* method [11]. Usually, a trajectory is taught to the robot by the well-known *walk-through* approach: the operator guides the robot by means of a teach-pendant through the sequence of desired arm positions [8]. These positions are then stored for posterior reproduction. This is a time-consuming approach and costly because the robot remains out of production during the teaching stage.

As the trajectory becomes more and more complex, with many intersecting via points, the operator may experience difficulties while setting up the correct temporal order of the trajectory points. This is the main motivation for the proposal of the neural model described in this paper, since it is highly desirable to have the teaching process with minimal human intervention. In the proposed approach, the responsibility of learning the temporal order of the trajectory is transferred to the neural network and, once training is completed, the stored trajectory can be used for autonomous trajectory planning, as depicted in Fig. 2.

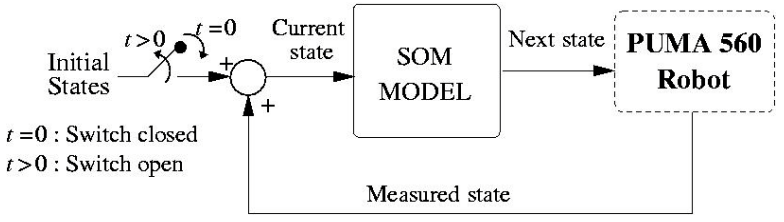


Fig. 2. Autonomous reproduction of a learned trajectory.

Another important issue is the role of the feedback path in Fig. 2, which allows the neural network to work autonomously, performing on a step-by-step basis the reproduction (planning) of the stored trajectory. This is important for safety purposes, since a trajectory only continues to be reproduced is the feedback pathway exists. Thus, if any problem occurs during the execution of the required motion by the robot, such collision with an obstacle or the joints reach their limiting values, the feedback pathway can be interrupted and the reproduction is automatically stopped.

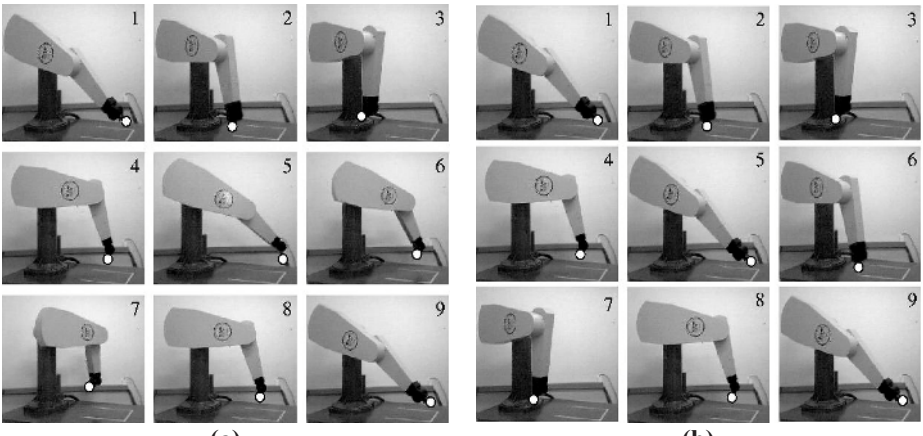
The conventional walk-through method does not possess the feedback pathway. In this case, all the trajectory states are sent to a memory buffer and executed in batch-mode. If any problem occurs, one has to wait for the execution of the whole trajectory in order to take a decision or to turn-off the robot power. Another important property of the VQTAM approach is its greater robustness to noise, as we show next through comparisons with a predictive variant of the lookup table method.

4 Tests with the VQTAM Model

The tests to be presented next evaluate the performance of the proposed method in the tasks of learning and reproduction (planning) of robot trajectories. The following issues will be assessed: accuracy of the retrieved trajectory, influence of the memory parameter (n_θ), influence of the type of neighborhood function, and tolerance of the model to noisy inputs. For this purpose, four trajectories whose pathways approximately describe an eight in 3D Euclidean space were generated by moving the robot through its workspace, each trajectory containing 9 states. This type of trajectory has been used as a benchmark for testing neural learning of robot trajectories because it has a repeated (crossing) via point. Thus, the reproduction of the stored trajectory states in the correct temporal order depends on temporal context information, which is represented by the parameter n_θ.

The proposed approach was implemented in C++ using the PUMA 560 control library QMOTOR/QRTK©, developed by Quality-Real Time Systems, running on a PC under the QNX real-time operating system. More details about the data acquisition processes and the interfacing hardware can be found at http://www.qrts.com and http://www.qnx.com.

The accuracy of the reproduction is evaluated by the *Normalized Root Mean Squared Error* (NRMSE), given by:



(a) **(b)**

Fig. 3. Influence of the memory on trajectory reproduction: (a) $n_\theta \geq 2$ and (b) $n_\theta = 1$.

$$NRMSE = \sqrt{\frac{1}{N_d.N_s}\sum_{t=1}^{N_d}\sum_{k=1}^{N_s}[\theta_i(k) - \hat{\theta}_i(k)]^2} \tag{14}$$

where N_d is the number of joints of the robot, N_s is the number of states of the trajectory, $\theta_i(k)$ is the desired value of joint i at time k and $\hat{\theta}_i(k)$ is the retrieved value of joint i at time k. The first test demonstrates the ability of the model in retrieving the stored trajectory states in the correct temporal order. We show only the results for one trajectory, since similar results are observed for the other three. A SOM model with 50 neurons was trained with the following parameters: $\eta_0 = 0.9$, $\sigma_0 = 25$, $n_\theta = 2$, $T = 5000$, $N_d = 6$ and $N_s = 9$. A correct reproduction is illustrated in Fig. 3a, where the number at the upper right corner of each subfigure denotes the position in time of that state of the robot arm and the open circle denote the position of the end-effector. An incorrect reproduction occurs if we use $n_\theta = 1$ (no memory!) as shown in Fig. 3b. In this case, the robot is unable to retrieve the whole trajectory, only half of it.

It is interesting to understand why the minimum value of n_θ is 2. This is equivalent to say that, to decide which route to follow, the neural network needs to have information about the current and the last state of the trajectory . This requirement can be easily understood if one notes that, to enter into one half of the trajectory the current state $\boldsymbol{\theta}(t)$ must be the crossing (bifurcation) via point and the last state $\boldsymbol{\theta}(t\text{-}1)$ must be in the other half of the trajectory. This is the minimum memory ``window´´ needed to reproduce the trajectory without ambiguity.

Fig. 4a shows the evolution of the NRMSE values as a function of memory parameter. It can be noted that for $n_\theta = 1$, the error is very high and that from $n_\theta \geq 2$ on, the error remains practically the same, confirming the result shown in Fig. 3b.

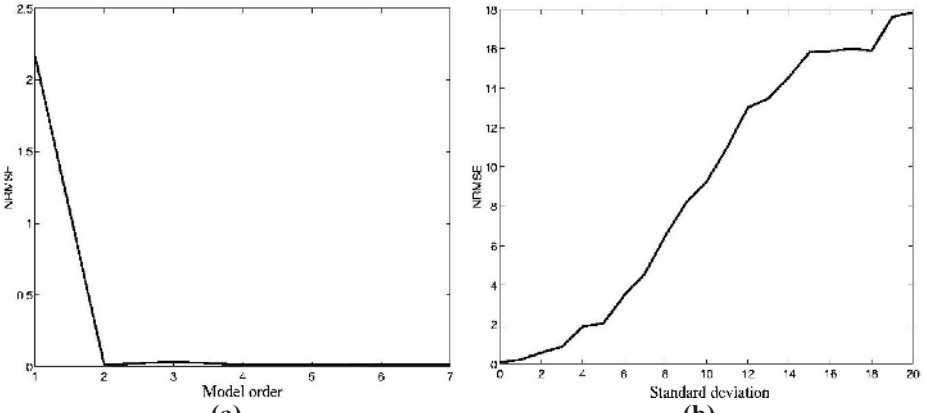


Fig. 4. (a) NRMSE versus memory order. (b) NRMSE versus noise variance.

The next set of tests evaluates the proposed model with respect to the presence of noise in the input vectors and the corresponding effects on the trajectory reproduction. Many tests were performed by adding Gaussian white noise, with zero mean and increasing variance σ^2, to the input vector. For each value of σ^2, we computed the NRMSE associated to the retrieved trajectory states, as shown in Fig. 4b.

One can note that the error increases gradually with the increase of the noise variance, even for very high values of σ^2, which is a direct consequence of the fact of using much more neurons than trajectory states. This occurs due to the topology preserving property of the SOM algorithm, i.e., neurons that are neighbors in the array have close (similar) weight vectors. Thus, a noisy input vector will be mapped onto neurons in the neighborhood of that neuron which would be the winner for the noise-free case. The resulting error is just slightly higher than the noise-free case.

The last test studies the influence of the choice of a neighborhood function $h(i^*, i; t)$ on the numerical accuracy of the reproduction. Two SOM networks were trained for different numbers of training epochs, one with a Gaussian and the other with a rectangular neighborhood function. The results are shown in Fig. 5. From Fig. 5a one can conclude that the rectangular neighborhood always provided lower values for the error. The effect on the retrieved joint angles is illustrated in Fig. 5b, where the retrieved angles of the robot base joint are shown for both types of neighborhood functions. From the exposed, it is recommended the use of the rectangular neighborhood function for two reasons, namely, (1) better accuracy and (2) lower computational cost.

Finally, we discuss the main differences of the model proposed in this paper and the CTH model. The CTH was the first entirely unsupervised neural algorithm applied to trajectory planning and point-to-point control of robotic manipulators, being tested by computer simulations in [2] and by implementation on a real PUMA 560 robot in [5]. The CTH has been applied to predictive modeling of robot trajectories, but it uses different learning mechanisms than those used by the VQTAM method.

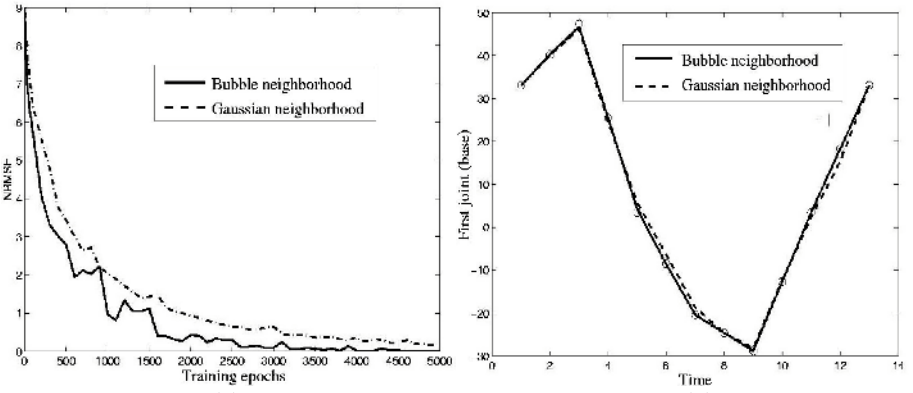


Fig. 5. (a) NRMSE versus training epochs. (b) Reproduction accuracy for different types of neighborhood functions.

The main differences are the following:

(i) The CTH uses two sets of weights to learn the temporal order of the states of a trajectory. The first set, comprised by feedforward competitive weights, stores the states, while the second, comprised by lateral Hebbian weights, learns the temporal transitions between consecutive states. The VQTAM needs only feedforward competitive weights to realize the same task.

(ii) The CTH algorithm has 6 training parameters to be determined by experimentation, while the VQTAM uses only two parameters: the learning rate and decay rate of the width of the neighborhood function.

(iii) The VQTAM is more tolerant to noise in the inputs because it benefits from the property of topology preservation of the SOM to produce errors that increase smoothly with the variance of the noise.

5 Conclusions

We introduced a self-supervised neural for prediction and planning of complex robot trajectories. The proposed technique extends Kohonen´s SOM so that it can learn dynamical input-output mappings for trajectory planning purposes. Several tests were carried out using a real PUMA 560 robot, aiming to evaluate the computational properties of the proposed model, such as robustness to noise, influence of the memory order on the model's performance, and the influence of the type of neighborhood function on the accuracy of the model. The obtained results have shown that the proposed approach performs better the current methods.

Acknowledgments. The authors thank CNPq (DCR:305275/2002-0) and FAPESP (Processes #00/12517-8 and #98/12699-7).

References

[1] Althöfer, K. and Bugmann, G. (1995). Planning and learning goal-directed sequences of robot arm movements, *in* F. Fogelman-Soulié and P. Gallinari (eds), *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, Vol. I, pp. 449–454.

[2] Araújo, A. F. R. and Barreto, G. A. (2002). A self-organizing context-based approach to tracking of multiple robot trajectories, *Applied Intelligence*, 17(1):99-116.

[3] Barreto, G. A. and Araújo, A. F. R. (2001). Time in self-organizing maps: An overview of models, *International Journal of ComputerResearch* **10**(2): 139–179.

[4] Barreto, G. A. and Araújo, A. F. R. (2002). Nonlinear modelling of dynamic systems with the self-organizing map, *Lecture Notes in Computer Science*, 2415:975-980.

[5] Barreto, G. A., , Dücker, C. and Ritter, H. (2002). A distributed robotic control system based on a temporal self-organizing network, *IEEE Transactions on Systems, Man, and Cybernetics-Part C* **32**(4): 347–357.

[6] Bugmann, G., Koay, K. L., Barlow, N., Phillips, M. and Rodney, D. (1998). Stable encoding of robot trajectories using normalised radial basis functions: Application to an autonomous wheelchair, *Proc. 29th Int. Symposium on Robotics (ISR)*, Birmingham, UK, pp. 232–235.

[7] Denham, M. J. and McCabe, S. L. (1995). Robot control using temporal sequence learning, *Proc. World Congress on Neural Networks (WCNN)*, Vol. II, Washington DC, pp. 346–349.

[8] Fu, K., Gonzalez, R. and Lee, C. (1987). *Robotics: Control, Sensing,Vision, and Intelligence*, McGraw-Hill.

[9] Hunt, K. J., Sbarbaro, D., Zbikowski, R. and Gawthrop, P. J. (1992). Neural networks for control systems – A survey, *Automatica* **28**(6): 1083–1112.

[10] Kohonen, T. (1997). *Self-Organizing Maps*, 2nd extended edn,Springer-Verlag, Berlin.

[11] Raibert, M. H. and Horn, B. K. P. (1978). Manipulator control using the configuration space method, *The Industrial Robot* **5**: 69–73.

[12] Ritter, H., Martinetz, T. and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps: An Introduction*, Addison-Wesley, Reading, MA.

[13] Walter, J. and Ritter, H. (1996). Rapid learning with parametrized self-organizing maps, *Neurocomputing* **12**: 131–153.