Competitive and Temporal Hebbian Learning for Production of Robot Trajectories

Guilherme de A. Barreto Aluizio F.R. Araújo
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

This paper proposes an unsupervised neural algorithm for trajectory production of a 6-DOF robotic arm. The model encodes these trajectories in a single training iteration by using competitive and temporal Hebbian learning rules and operates by producing the current and the next position for the robotic arm. In this paper we will focus on trajectories with one common point. These types of trajectories introduce some ambiguities, but even so, the neural algorithm is able to reproduce them accurately and unambiguously due to context units used as part of the input. In addition, the proposed model is shown to be fault-tolerant.

1. Introduction

One of the major aspects of natural intelligence is the ability to process temporal information. Temporal pattern learning and recalling are crucial for our capacity to perceive and generate limb movements, speech, music, etc. In addition, because we live in an environment that changes continuously, an intelligent system must be able to encode patterns over time and to reproduce them [1].

During the last years, many neural network approaches have been proposed for the trajectory generation problem. One of them is known as *robot trajectory learning* or *robot trajectory encoding* [2],[3].

According to the approach above, an artificial neural network receives as input the current state (such as position, joint angles and associated torques) of the robot arm and usually responds with the next state in order to execute a task defined in advance. This anticipatory behavior is particularly useful for solving trajectory ambiguities [4] and is suitable for point-to-point trajectory control or for trajectory tracking.

Regarding the encoding of robot trajectories, Althöfer and Bugmann [2] described two types of neural networks for learning and planning robot arm movements. The first one, a neural implementation of a resistive grid [5] for path planning, presented limitations such as jerkiness of the movements and inaccurate final positions. The second network is trained during the read-out of a sequence of movements as determined by the resistive grid. This network solves the former limitations by using a RBF model with receptive fields centered on a sequence of starting positions in the configuration space of the arm, and with weights to the output layer being used to point out to the next position in the space. This network generates a smooth arm trajectory and an accurate positioning by interpolating points between those given by means of the resistive grid network.

In the context of mobile robotics, Bugmann et al. [3] proposed a model that uses Normalized Radial Basis Functions to encode a sequence of positions forming the trajectory of an autonomous wheelchair. The network produces the next position for the wheelchair. As the trajectory passes several times over the same point, phase information is added to the position information to avoid the *perceptual aliasing*[1] problem [4]. The use of Normalized RBF's creates an attraction field over the whole workspace and enables the wheelchair to handle perturbations caused by the avoidance of people.

Araújo and Vieira [6] and Araújo e D'arbo Jr. [7] proposed, respectively, temporal associative memory and recurrent neural networks models for learning and production of trajectories of a 6-DOF robot arm. In both cases, the initial and goal positions are given and the networks produce all the intermediate positions and its associated joint angles and torques. Both models are able to retrieve the trained trajectories and are robust (to a certain degree) to noise. The simulations suggested that the recurrent neural model is suitable for tracking the trajectory while the associative memory model is adequate for interpolating trajectory states.

[1] This problem involves two or more identical perceptual inputs that require different responses from the robot.

In this paper, the main goal is to devise an unsupervised neural algorithm to learn and retrieve temporal sequences in the form of robot trajectories in order to perform tracking tasks.

The remaining part of the paper is organized as follows. In Section 2, we present the proposed architecture for the unsupervised network and discuss each component. In Section 3, we introduce the proposed algorithm and discuss its dynamics in details. In Section 4, some simulations are carried out in order to show the model ability to learn and reproduce the desired trajectories. The fault-tolerance is also evaluated. Finally, in Section 5 we conclude the paper.

2. The Proposed Architecture

The set of points that specifies the translational and rotational paths of the manipulator end-effector as a function of time is referred to as a trajectory [8]. In our case, the robot trajectories are specified as sequences of eleven points (or states) in which each state is a 15-dimensional vector consisting of the Cartesian coordinates (x, y, z), the joint angles (θ_1,..., θ_6), and the torques (τ_1,..., τ_6) associated with that spatial position.

The architecture of the proposed model is shown in Figure 1. It comprises two layers of neurons, in which each input node is connected to all output neurons through feedforward weights w_{ji}, where i indicates the input node and j, the output one. The network activates the neurons that encode the current and the next states of the trajectory once an input state arrives.

Every time an input state is presented to the net, a neuron with the most similar weight vector to the input is activated. This neuron is called the winner of the competition for that input pattern. The weights, arriving to this particular winning neuron, encode the state of the trajectory at a particular instant of time.

In this model, each state of the trajectory must be represented by a single neuron (or a small group of neurons). Thus, a mechanism must be provided to avoid that such a group responds for more than one state of the trajectory. The mechanism adopted in the current work, exclude the current winning neuron from all subsequent competitions for trajectory states. Furthermore, the exclusion mechanism allocates different neurons for equal states occurring in different instants of time. This kind of situation may occur when a trajectory passes more than one time through the same point.

Each output neuron projects a connection to itself, $m_{jj}(t)$, and to all other nodes, $m_{jr}(t)$. These connections encode the temporal ordering of the input trajectory. Once active, the current winning neuron triggers, through the lateral connections, the neuron that encodes the next point of the trajectory. Both neurons remain active. In order to differentiate the responses, the activation of the current winning neuron is made lower (through the self-connection) than the activation of the neuron encoding the next state.

It is important to note the presence of time delays in the proposed architecture. They implement a short-term memory (STM) model [1]. That enables temporal association between patterns occurring in consecutive instants of time.

The context vector is of great importance to the proposed architecture. Without it, only trajectories with no points in common can be reproduced. The context information allows the network to distinguish between equal states belonging to different trajectories. For example, two trajectories with a state in common usually have different initial or goal states. One of these can be used as a context information.

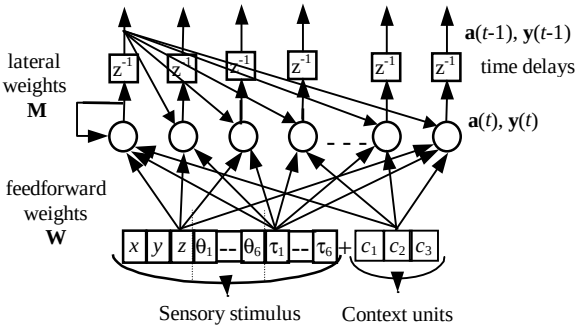


Figure 1. The topology of the proposed network, where the time delays implement the short-term memory. Only some connections are shown.

Differently from previous models for trajectory encoding [3], [4], which use pre-wired network weights, the proposed algorithm learns to encode the temporal order by self-organization. In the next section the neural algorithm is presented and discussed in detail.

3. The Proposed Algorithm

The steps of the neural algorithm are defined as:

(1) Initialize the network as follows:

$$w_{ji}(0) = \text{random}[0, 1], \text{ for all } i, j;$$

$$m_{jr}(0) = 0, \text{ for all } j, r;$$

$$a_j(0) = y_j(0) = 0, \text{ for all } j.$$

$$f_j(0) = 1, \text{ for all } j.$$

where random[0, 1] is a number between 0 and 1. $a_j(t)$ and $y_j(t)$ are the activation and output of neuron j, respectively. The function $f_j(t)$ is called neuron exclusion factor.

(2) Present an input stimulus to the net.

(3) Determine the winning neurons:

For every input vector, order the output neurons according to their distance to the input vector $\mathbf{v}(t)$. i.e:

$$f_{\mu_1}(t)\left\|\mathbf{v}(t)-\mathbf{w}_{\mu_1}(t)\right\| < f_{\mu_2}(t)\left\|\mathbf{v}(t)-\mathbf{w}_{\mu_2}(t)\right\| <\cdots$$

$$\cdots< f_{\mu_k}(t)\left\|\mathbf{v}(t)-\mathbf{w}_{\mu_k}(t)\right\| <\cdots< f_{\mu_N}(t)\left\|\mathbf{v}(t)-\mathbf{w}_{\mu_N}(t)\right\| \qquad (1)$$

where $\mu_1(t)$ is the index of the winning neuron (the one closest to the input) of the current competition, $\mu_2(t)$ is the index of the second neuron closest to the input, and so on. The index k indicates the number of neurons used to encode each input pattern per competition (for example, for k=2, two neurons will remain active) and N is the number of output neurons. The choice of more than one neuron per state prevents catastrophic loss of a entire trajectory when a single neuron fails.

The function $f_j(t)$ is updated according to:

$$f_j(t)=\begin{cases}\alpha, & \text{for } j=\mu_1,\mu_2,\ldots,\mu_k\\ 1, & \text{Otherwise.}\end{cases} \qquad (2)$$

where $\alpha >> 0$ is large enough in order to exclude the chosen neurons from all subsequent competitions.

(4) Determine the activations:

The activations are determined by

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

and A and γ are constant values, so that $A > \gamma$.

(5) Adjust the feedforward weights:

Adapt the feedforward weight vectors according to the following competitive learning rule [9]:

$$\mathbf{w}_j(t+1)=\mathbf{w}_j(t)+\delta(t)\,a_j(t)\left[\mathbf{x}(t)-\mathbf{w}_j(t)\right] \qquad (4)$$

where δ ($\approx$1) is the learning rate determining how similar to the input $\mathbf{x}(t)$ will be the weight vector $\mathbf{w}_j(t)$. Only those neurons with activations not equal to zero are allowed to learn.

(6) Adjust the self- and lateral connections:

The lateral weights are liable for encoding the temporal ordering of the input sequence. When an input stimulus is presented, the network will activate at least two neurons. One should respond for the current input pattern (less activated neuron) and the other indicates the next state of the trajectory (most activated neuron).

The ideas behind the learning rules are the following: (i) self-connections should weaken the activation, and (ii) lateral connections from the winners of the last competition to the winner of the current competition should be Established. By inhibiting its activity through the self-connection and exciting other neuron through a lateral weight, the winning neuron of the current competition determines the current and the next state of the trajectory. Mathematically, we have the following learning rules:

(6a) Learning procedure for self-connections:

for $\{j = \mu_1(t), \mu_2(t), \ldots, \mu_k(t)\}$ do:

$$m_{jj}(t+1)=m_{jj}(t)+\beta a_j(t), \qquad (5)$$

where β is the learning rate and $\mu_1(t)$, $\mu_2(t)$, ..., $\mu_k(t)$ are the indexes of the winners of the current competition.

(6b) Learning procedure for lateral connections:

for $\{j = \mu_1(t), \mu_2(t), \ldots, \mu_k(t)\}$,

for $\{r = \mu_1(t\text{-}1), \mu_2(t\text{-}1), \ldots, \mu_k(t\text{-}1)\}$ do:

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

where λ=1-β is the learning rate, $\mu_1(t)$, $\mu_2(t)$, ..., $\mu_k(t)$ are the winners of the current competition and $\mu_1(t\text{-}1)$, $\mu_2(t\text{-}1)$, ..., $\mu_k(t\text{-}1)$ are the winners of the last competition. Note that equation (6) represents a simple temporal version of the correlation (or Hebbian) learning [10] between the activations of the output neurons in different instants of time. Thus, connections are always established from the winners at time t-1 (represented by the index r) to winners at time t (represented by the index j). By defining $\beta<\lambda$, the activation of the neuron representing the current input is made smaller than that of the neuron encoding the next state of the trajectory. The activations $a_r(t\text{-}1)$ in (6) are obtained through the time-delays.

(7) Determine the outputs:

The outputs are determined through the simple weighted summation:

$$y_j(t) = \sum_{r=1}^{N} m_{jr}(t)\, a_r(t), \quad (7)$$

where $m_{jr}(t)$ is the connection weight between the output neurons r and j.

Suppose that neuron three is the winner for the current input state, so according to equations (1) and (3) we have: $\mu_1(t) = 3$ and $a_3(t) = 1$. The remaining activations are zero (we have chosen k=1, for simplicity). Now, also suppose that the next state of the trajectory was encoded by neuron 7. Hence, because the temporal order is to be encoded by the lateral weights, there must be a non-zero self-connection for neuron 3 and a non-zero lateral connection from neuron 3 to neuron 7. Thus, m_{33}=0.2 and m_{73}=0.8, for example. Thus, according to equation (7) the outputs during the reproduction phase are:

$$y_3(t) = m_{33}(t)a_3(t) = (0.2).(1.0) = 0.2 \quad (8)$$

$$y_7(t) = m_{73}(t)a_3(t) = (0.8).(1.0) = 0.8 \quad (9)$$

$$y_j(t) = 0, \text{ for all } j \neq 3 \text{ and } j \neq 7 \quad (10)$$

Equations (8), (9) e (10) mean that the neuron 3 is the most similar to the current input pattern and neuron 7 is the next state of the trajectory. The process continues from step (2) until the end of the trajectory is reached. For trajectory reproduction, steps (5) and (6) are skipped. In the next section, we show some simulations results involving different trajectories.

4. The simulations

The different trajectories considered for study are shown in Figure 2.

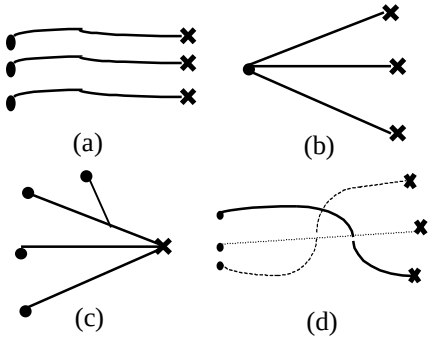


Figure 2. Types of trajectories considered for study. The filled circle indicates the initial position and "x" stands for the final position.

The trajectories 2b, 2c, and 2d have at least one point in common, which may introduce ambiguities during the reproduction of the sequence. This problem is stated as: which trajectory should the arm follow when each candidate has points in common with another one? This problem is solved by using a fixed context vector given as part of the input, remaining clamped to the input during both trajectory learning and reproduction phases.

In this paper, the trajectories were defined in advance, being generated by the toolbox robotics of MATLAB [11]. However, the robot arm could be trained by a "teach-by-showing" method. In this case, an user would push the arm through the desired trajectories.

The network parameters were set to α = 1000, k=2, A = 1, γ = 0.97, δ = 0.99, β = 0.2, and λ = 1-β = 0.8. The number of output neurons was set to 70 and three trajectories per case were trained sequentially in only one presentation for each trajectory. The context units were made equal to the goal position of the trajectory under consideration, changing when another trajectory is considered. The tests are similar to those presented in [6] and [7], and for Figures 3-6 only the results for the first winner are shown.

As a measure for the tracking error, we used the following expression:

$$E_t = \frac{1}{N_p} \sum_{t=1}^{N_p} \left\{ \left(x_d^t - x_r^t\right)^2 + \left(y_d^t - y_r^t\right)^2 + \left(z_d^t - z_r^t\right)^2 \right\} \quad (11)$$

where (x_d, y_d, z_d) and (x_r, y_r, z_r) are the desired and the retrieved spatial coordinates, N_p is the number of points of the trajectory and t is the position in the sequence. For example, t=1 indicates the first vector in the sequence.

Figure 3 shows the results for three trajectories with the same starting point located at (0.6, 0.1, 0.0). Note that the retrieved and the desired trajectories in all the three cases are very similar. The tracking error for trajectory **I-G1** is 5.224×10^{-5}. This illustrates the ability of the model in accurately encoding an input state in only one iteration.

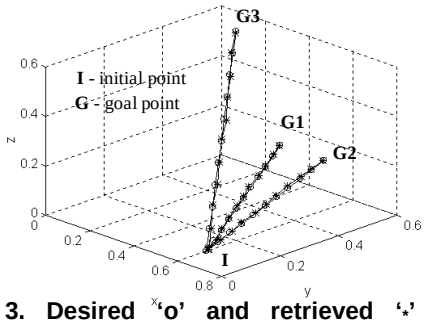


Figure 3. Desired 'o' and retrieved '*' spatial trajectories with same starting point (0.6, 0.1, 0.0).

Figure 4 illustrates the desired and the retrieved joint angles for the trajectory **I-G1**. It also has very good performance, confirming the adequacy of the model for trajectory tracking. The ambiguity faced by the network when it has to decide which trajectory to follow is resolved by the knowledge of the goal position.

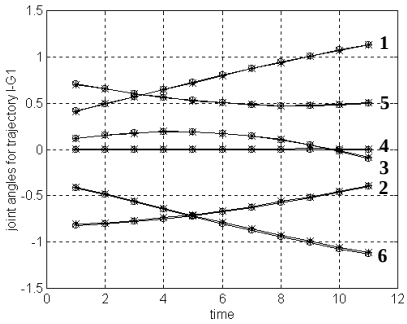
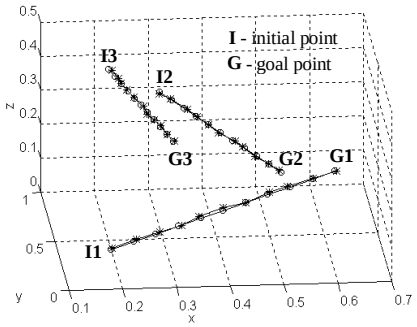


Figure 4. The desired 'o' and the retrieved '*' joint angles for trajectory I-G1 in Figure 3.

Figure 5 illustrates the results for three trajectories with no points in common. This situation is considered by the model as the easiest one to be encoded, because there is no ambiguity. The model was able to reproduce the trained trajectories with a small tracking error (e.g. 4.67×10^{-5}, for the sequence I1-G1).

(a)



(b)

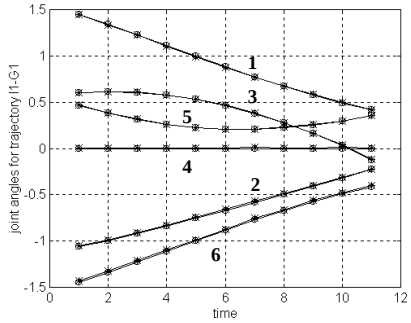
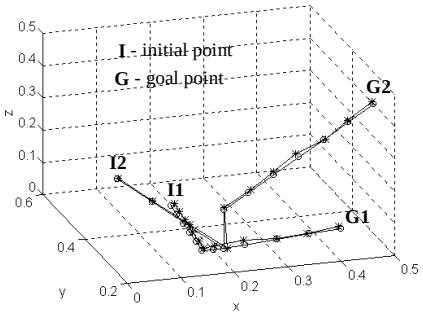


Figure 5. Trajectories without common points: (a) spatial points; (b) joint angles and (c) for trajectory I1-G1.

The next simulation considers trajectories with one crossing point. The desired and retrieved spatial position are shown in Figure 6a. The corresponding joint angles are shown in Figure 6b. In this case, the context units also play an important role. The trajectories are harder to be followed because of their abrupt change of directions. Even so, the model was able to track them with a small error (6.92×10^{-5} for trajectory I1-G1).

(a)



(b)

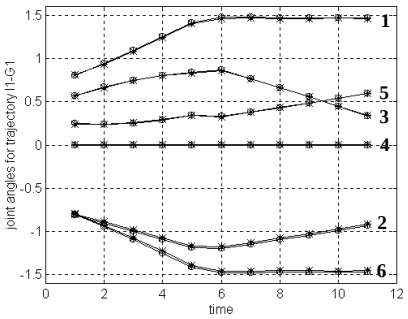
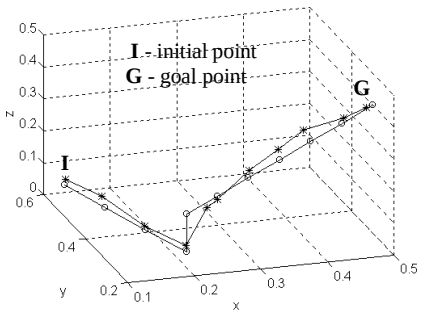


Figure 6. Trajectories with one crossing point: (a) spatial positions: (b) joint angles for trajectory I1-

The last simulation explores the fault-tolerance ability of the model. We have simulated a worst-case situation, namely: all first winners for each state of the trajectory I2-G2 in Figure 5a have collapsed for some reason (for example, hardware failure in a real implementation). The results are shown in Figure 7.

(a)



(b)

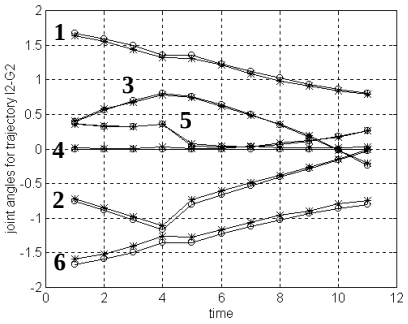


Figure 7. Simulation of the worst-case condition (collapse of all first winners) for trajectory I2-G2 in Figure 6a: (a) spatial points and (b) joint angles.

Even for this catastrophic situation, the tracking error was quite small, 5.91×10^{-4}. It is worth noting in Figure 7a that the network dynamics tend to maintain an accurate initial and final position even for second winners. The joint angles are shown in Figure 7b.

One could ask that this last result can be improved if we change step 6 of the algorithm by assigning the same activation value to all winners (a_j=1 for all j=1..k). In this case we would be improving the fault-tolerance and the total accuracy but worsening the tolerance to noise and the generalization ability of the proposed model.

Table 1 summarizes the tracking errors for the trajectories discussed in this work. The best condition refers to the situation in which the first winners for each state of the trajectories are used to retrieve them.

Table 1. Summary of tracking errors

TRAJECTORY TYPE	BEST CONDITION	WORST CONDITION
common initial point (Figure 3)	I-G1: 5.2213×10^{-5} I-G2: 6.7044×10^{-5} I-G3: 4.1415×10^{-5}	I-G1: 5.9863×10^{-4} I-G2: 6.9996×10^{-4} I-G3: 5.9260×10^{-4}
common final point (not shown)	I1-G: 5.7652×10^{-5} I2-G: 3.5489×10^{-5} I3-G: 3.7065×10^{-5}	I1-G: 7.1104×10^{-4} I2-G: 7.7006×10^{-4} I3-G: 6.3143×10^{-4}
without common points (Figure 5)	I1-G1: 4.6685×10^{-5} I2-G2: 3.4346×10^{-5} I3-G3: 2.8334×10^{-5}	I1-G1: 9.2189×10^{-4} I2-G2: 7.2673×10^{-4} I3-G3: 6.4547×10^{-4}
common intermediate point (Figure 6)	I1-G1: 6.9224×10^{-5} I2-G2: 4.9082×10^{-5} I3-G3: 4.8149×10^{-5}	I1-G1: 1.0087×10^{-3} I2-G2: 8.6517×10^{-4} I3-G3: 5.9096×10^{-4}

The results for the torques associated with joint angles were not shown because of lack of space, but the proposed network model was able to track them with small error as well.

5. Conclusions and Further Work

In this paper, we have proposed a temporal sequence based control system for robotic trajectory learning. Despite the simplicity of the model, the system has a combination of properties which are of great importance for the design of intelligent robotic systems, namely:

(1) Accurate recall of stored temporal patterns;
(2) Disambiguation when trajectories have points in common;
(3) Tolerance to faults, since a sequence can still be retrieved even in the presence of neuronal failure;
(4) Simple and fast learning;
(5) Learning of inverse kinematics and inverse dynamics as well;
(6) Lower computational cost when compared with supervised learning; and
(7) The algorithm can be potentially adapted to work on others temporal sequence tasks (such as mobile robot control, speech recognition, and natural language processing).

Further work must be developed in order to explore the generalization ability of the proposed model and extend the present model to one which generates appropriate control actions over the whole robot working space given a number of trajectories to be learned.

Acknowledgments

The authors would like to thank PICDT/CAPES and FAPESP for financial support.

6. References

[1] D.-L. Wang. “Temporal pattern processing”. In: The Handbook of Brain Theory and Neural Networks, M.A. Arbib (ed.), MIT Press, 967-971, 1995.

[2] K. Althöfer and G. Bugmann. “Planning and learning goal-directed sequences of robot arm movements”. In: Fogelman-Soulié F. and Gallinari P. (eds.), Proc. of the Intl. Conference on Artificial Neural Networks, Paris, France, 1:449-454, 1995.

[3] G. Bugmann, K.L. Koay, N. Barlow, M. Phillips, and D. Rodney. “Stable encoding of robot trajectories using normalized radial basis functions: application to an autonomous wheelchair”. Proc. of the 29th International Symposium on Robotics, Birmingham, UK, April 1998.

[4] R.P.N. Rao and O. Fuentes. “Learning navigational behaviors using a predictive sparse distributed memory”. Proc. of From Animals to Animats: the 4th Intl. Conf. on Simulation of Adaptive Behavior, MIT Press, 382-390, 1996.

[5] P. Musílek. “Neural networks in navigation of mobile robots: A survey”, *Neural Network World*, 6:929-943, 1995.

[6] A.F.R. Araújo and M. Vieira. “Associative memory used for trajectory generation and inverse kinematics problem”. Proceedings of the IEEE World Congress on Computational Intelligence, Anchorage, USA, 2057-2052, May 1998.

[7] A.F.R. Araújo and H. D’Arbo Jr. “Partially recurrent neural network to perform trajectory planning, inverse kinematics, and inverse dynamics”. Paper accepted for presentation in the IEEE International Conference on System, Man, and Cybernetics, San Diego, USA, October 1998.

[8] A.J. Koivo. “Fundamentals for Control of Robotics Manipulators”. John Wiley & Sons, Inc., New York, 1989.

[9] J. Hertz, A. Krogh, and R.G. Palmer. “Introduction to the theory of neural computation”. Addison-Wesley, 1991.

[10] D.O. Hebb. “The organization of behavior”. New York: Wiley, 1949.

[11] P.I. Corke. “A robotics toolbox for MATLAB”. *IEEE Robotics and Automation Magazine*, 3(1):24-32, 1996.