ELSEVIER

Artificial Intelligence in Engineering 13 (1999) 393–398

ARTIFICIAL INTELLIGENCE in ENGINEERING

www.elsevier.com/locate/aieng

# A robotic system based on neural network controllers

L. Acosta*, G.N. Marichal, L. Moreno, J.J. Rodrigo, A. Hamilton, J.A. Mendez

*Department of Applied Physics, University of La Laguna, La Laguna 38271 Tenerife, Spain*

**Abstract**

In this paper, a control algorithm based on neural networks is presented. This control algorithm has been applied to a robot arm which has a highly nonlinear structure. The model based approaches for robot control (such as the computed torque technique) require high computational time and can result in a poor control performance, if the specific model-structure selected does not properly reflect all the dynamics. The control technique proposed here has provided satisfactory results. A decentralised model has been assumed here where a controller is associated with each joint and a separate neural network is used to adjust the parameters of each controller. Neural networks have been used to adjust the parameters of the controllers, being the outputs of the neural networks, the control parameters. © 1999 Elsevier Science Ltd. All rights reserved.

*Keywords:* Neural networks; Control; Robotics

## 1. Introduction

As it is well known the dynamics of a robot arm [1] can be expressed by the following general set of equations:

$$\sum_j d_{kj}(q)\ddot{q}_j + \sum_{i,j} c_{ijk}(q)\dot{q}_i\dot{q}_j + f_k(q) = \tau_k \qquad (1)$$

$$k = 1,\ldots,n,\ i = 1,\ldots,n,\ j = 1,\ldots,n$$

where

| | |
|---|---|
| $q_j$ | $j$th generalized co-ordinate. |
| $q$ | generalized co-ordinate vector. |
| $\tau_k$ | $k$th generalized force. |
| $n$ | number of joints. |
| $d_{kj}$ | inertial coefficients. |
| $c_{ijk}$ | centrifugal and Coriolis coefficients. |
| $f_k$ | loading item due to gravity. |

In the case of a PUMA robot, the generalized coordinates are the joint angles and the generalized forces are the joint torques. In Fig. 1, a photograph of the PUMA robot is shown. As can be seen in Fig. 2, it has five degrees of freedom with different range of angles corresponding to each joint. Notice that there is a torque associated with each joint, such that the movements of the links are caused by these torques. In the following, all these torques will be considered as the components of the torque vector associated to the robot.

The highly non-linear and coupled dynamics of robot arms [2–4], as well as the often unknown inertial properties make controlling this kind of systems difficult. Several approaches based on different kinds of neural networks have been proposed to control highly nonlinear systems [5,6]. Here, it is worth mentioning the control algorithm proposed by Kawato et al. [7] termed feedback error learning. This control algorithm is based on using feedback controllers and neural networks applied to controlling a robot arm. The feedback controllers provide the values of the torque, whereas the neural networks modify these values by adding some values to the torque before they are applied to the joints. In this manner, the neural networks modify the values of the torque given by the feedback controller in order to have a better performance of the control algorithm. In this paper an alternative computational approach is presented where the applied values of the torque are calculated directly by the feedback controllers, while the neural networks provide the parameters of the feedback controllers.

## 2. PUMA robot control

In order to control the position of the links, neural network controllers have been used [8,9]. In our approach, there is no assumption about a parametric model for the robot. The neural networks provide the parameters for the controllers [10]. Concretely, a proportional derivative controller and a proportional integral derivative controller

* Corresponding author. Tel.: + 34-22-635475; fax: + 34-22-603217.
*E-mail address:* nico@cyc.dfis.ull.es (L. Acosta)

0954-1810/99/$ - see front matter © 1999 Elsevier Science Ltd. All rights reserved.
PII: S0954-1810(99)00012-6

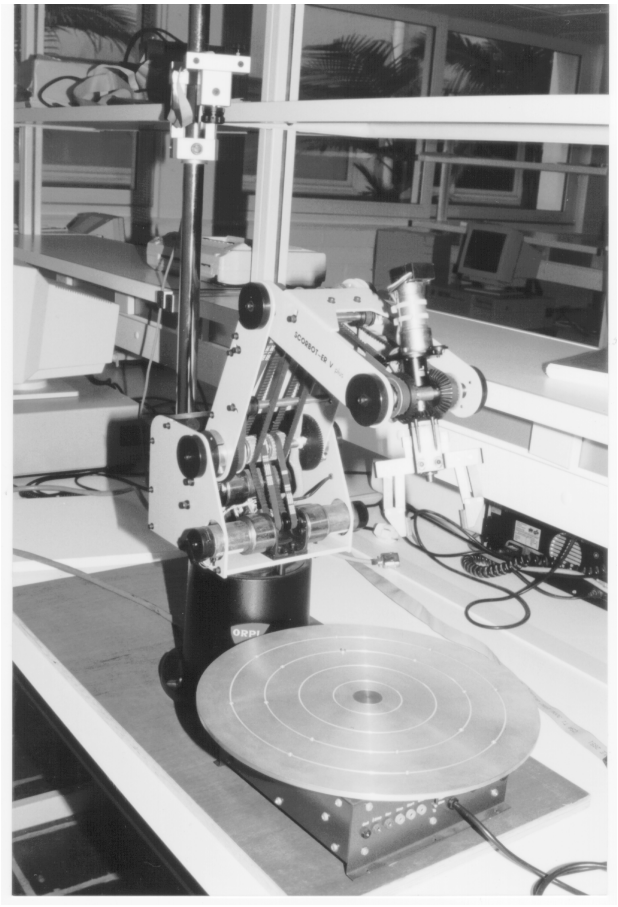

Fig. 1. Robot PUMA.

have been used, where the corresponding parameters are the outputs of the neural networks. In Fig. 3, a general sketch of this control approach is shown. Note that, $(\theta_{d1}, \theta_{d2}, \ldots, \theta_{dn})$ refers to the desired joint angles for each joint, while $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$ refers to the current joint angles of the PUMA robot.

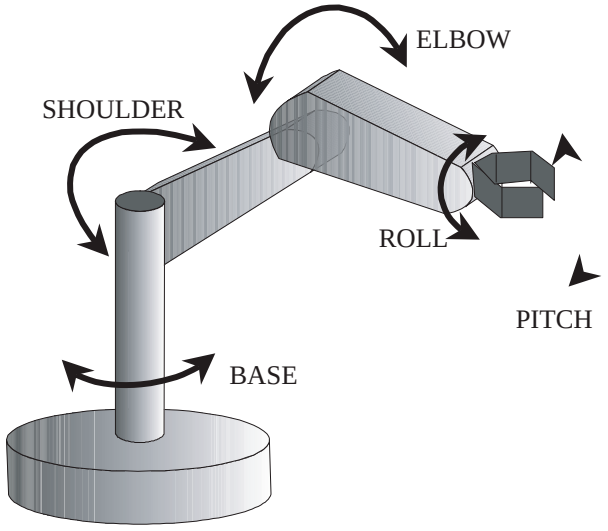


Fig. 2. Sketch of the PUMA.

It is important to point out that a decentralised model has been assumed for control structure. That is, each controller determines one joint torque. Furthermore, the same kind of controller has been considered for all the links. However, the neural networks have used all the joint angles and joint velocities in order to determine the parameters corresponding to each controller.

In Fig. 3, the inputs to the neural network are the current joint robot angles and their angular velocities. On the other hand, the outputs are the parameters of the controllers, that is, the proportional gain and the derivative gain if proportional derivative controllers are used or the proportional gain, the derivative gain and the integral gain if proportional integral derivative controllers are used.

As a learning algorithm a backpropagation algorithm has been chosen [11,12]. The following notation is used:

| | |
|---|---|
| $v_{l,j}$ | output of the $j$th node in layer $l$; |
| $w_{l,j,i}$ | weight which connects the output of the $i$th node in layer $l$-1 to the input of the $j$th node in layer $l$; |
| $\mathbf{x}$ | training sample; |
| $d_j(\mathbf{x})$ | desired response of the $j$th output; |
| $N_l$ | number of nodes in layer $l$; |
| $L$ | number of layers. |

The output of a node is given by

$$v_{l,j} = f\left(\sum_{q=1}^{N_{l-1}} w_{l,j,i} v_{l-1,i}\right) \tag{2}$$

where $f(\cdot)$ is the sigmoid function.

The objective is to change the weights of the network in order to minimize an objective function of the form:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{N_L}\left(d_q(\mathbf{x}) - v_{L,q}(\mathbf{x})\right)^2 \tag{3}$$

The backpropagation technique states that the weights are updated in each iteration according to

$$w_{l,j,i}(k+1) = w_{l,j,i}(k) - \mu\frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}} \tag{4}$$

where $\mu$ is the *learning rate*. It can be shown that the sensitivity function is

$$\frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}} = \frac{\partial J(\mathbf{w})}{\partial v_{l,j}}\frac{\partial v_{l,j}}{\partial w_{l,j,i}}; \tag{5}$$

$$\frac{\partial J(\mathbf{w})}{\partial v_{l,j}} = \sum_{m=1}^{N_{l+1}}\frac{\partial J(\mathbf{w})}{\partial v_{l+1,m}}(v_{l,j}(1 - v_{l+1,m})w_{l+1,m,j}); \tag{6}$$

$$\frac{\partial v_{l,j}}{\partial w_{l,j,i}} = (v_{l,j}(1 - v_{l,j})v_{l-1,i}); \tag{7}$$

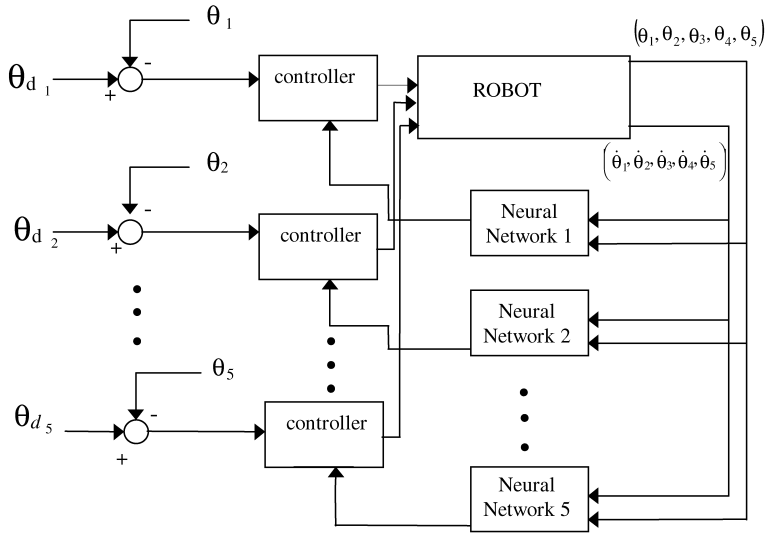


Fig. 3. Sketch of the control system.

with

$$\frac{\partial J}{\partial v_{L,j}} = v_{L,j}(\mathbf{x}) - d_j(\mathbf{x}) \tag{8}$$

The backpropagation algorithm can be summarised as follows:

1. Initialise weights to small random values.
2. Propagate the input signal forward throughout the network.
3. Compute the sensitivity function for each weight of the network.
4. Update weights.
5. Go to step 2 and repeat the procedure until end condition is reached.

In the case presented in this paper the criterion function $J(\mathbf{w})$ has been taken as follows:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{N}(\theta_{dq} - \theta_q)^2 \tag{9}$$

where

$\theta_{dq}$ desired joint angle corresponding to $q$th link
$\theta_q$ joint angle corresponding to $q$th link
$N$ number of robot links.

In the subsequent development, only one link has been considered link $h$, since a decentralised model has been chosen. Only one of the neural networks will be analysed, the expressions for the others will be identical.

The process for updating the weights is done according to (4). Then, it is necessary to calculate the term $\partial J(k)/\partial w_{l,j,i}$. If the output of the neural network were directly the variable $\theta_j$, this derivative for the output layer would be:

$$\frac{\partial J(k)}{\partial w_{L,j,i}} = \frac{\partial J(k)}{\partial \theta_j(k)}\frac{\partial \theta_j(k)}{\partial w_{L,j,i}} = -e_j(k)\frac{\partial \theta_j(k)}{\partial w_{L,j,i}} \tag{10}$$

$e_j(k)$ being the output error corresponding to the $j$th joint angle. In the controller proposed here, the outputs of the neural network $m_j$ (the different parameters of the controller) do not appear in the criterion function explicitly. So the sensitivity function becomes:

$$\frac{\partial J(k)}{\partial w_{Lji}} = \frac{\partial J(k)}{\partial m_j}\frac{\partial m_j}{\partial w_{l,j,i}} \tag{11}$$

where

$$\frac{\partial J(k)}{\partial m_j} = -\sum_{i}^{N} e_i(k)\frac{\partial \theta_i(k)}{\partial \tau(k)}\frac{\partial \tau(k)}{\partial m_j} \tag{12}$$

Note that $\tau$ denotes the joint torque applied to the particular link $h$ treated in this section. The factor $\partial\theta_i(k)/\partial\tau(k)$ is obtained immediately if the model of the robot is known. Avoiding any assumption about the robot model, this term can be obtained by difference approximation:

$$\frac{\partial \theta_i(k)}{\partial \tau(k)} \approx \frac{\theta_i(k) - \theta_i(k-1)}{\tau(k) - \tau(k-1)} \tag{13}$$

The factor $\partial\tau(k)/\partial m_j$ is calculated from the expression for the controller. If the controller is changed, it is only necessary to change the sensitivity function, and the rest of the algorithm remains the same.

In the case where a proportional derivative controller were used, the expression for the torque would be the following:

$$\tau_h = m_1(\theta_{dh} - \theta_h) - m_2\dot{\theta}_h \tag{14}$$

where

$m_1$ proportional gain

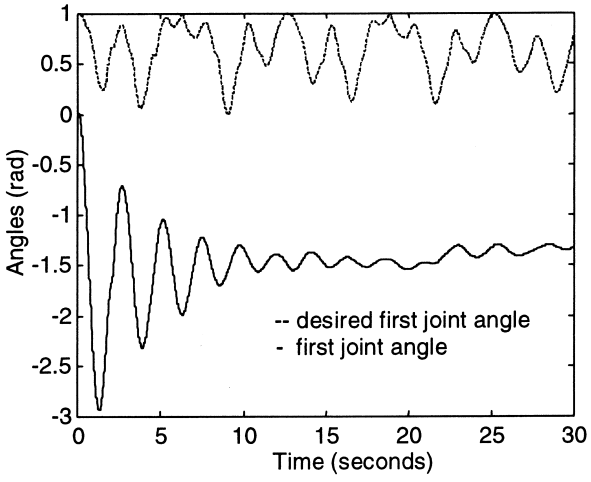


Fig. 4. Joint angles vs. time: first try.

$m_2$ derivative gain.
$\theta_{d1}$ desired joint angle corresponding to first joint.
$\theta_1$ joint angle corresponding to first joint.

Note that $m_1$ and $m_2$ should be named $m_{1h}$ and $m_{2h}$ given that the parameters are different for each link, but for the sake of simplicity the index $h$ will be eliminated.

Therefore, calculating the partial derivatives of the torque with respect to the parameters, we get:

$$\frac{\partial \tau_h}{\partial m_1} = (\theta_{dh} - \theta_h); \tag{15}$$

$$\frac{\partial \tau_h}{\partial m_2} = -\dot{\theta}_h. \tag{16}$$

On the other hand, it is necessary to calculate the partial derivatives of the criterion function with respect to the parameters in order to obtain the sensitivity function for the weights of the output layer. It can be calculated using Eq. (12) as:

$$\frac{\partial J(k)}{\partial m_1} = -\sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} (\theta_{dh} - \theta_h) \tag{17}$$

$$\frac{\partial J(k)}{\partial m_2} = \sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} \dot{\theta}_h \tag{18}$$

And finally the sensitivity function $\partial J(k)/\partial w_{L,j,i}$ for the weights of the output layer are calculated as follows:

$$\frac{\partial J(k)}{\partial w_{L,1,s}} = -\sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} (\theta_{dh} - \theta_h) \frac{\partial m_1}{\partial w_{L,1,s}} \tag{19}$$

$$\frac{\partial J(k)}{\partial w_{L,2,s}} = \sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau(k) - \tau(k-1)} \dot{\theta}_1 \frac{\partial m_2}{\partial w_{L,2,s}} \tag{20}$$

where $\partial m_1/\partial w_{L,1,s}$ and $\partial m_2/\partial w_{L,2,s}$ are by the Eq. (7) as:

$$\frac{\partial m_1}{\partial w_{L,1,s}} = (m_1(1 - m_1) v_{L-1,s}); \tag{21}$$

$$\frac{\partial m_2}{\partial w_{L,2,s}} = (m_2(1 - m_2) v_{L-1,s}). \tag{22}$$

The new weights of the output layer can be updated using the sensitivity functions given by Eqs. (19) and (20), where all terms are known. On the contrary, the update of the rest of weights would be done as usual, using the Eqs. (5)–(7).

If a proportional integral derivative controller were used, the torque would be:

$$\tau_h = m_1(\theta_{dh} - \theta_h) - m_2 \dot{\theta}_h + m_3 \int_0^t (\theta_{dh} - \theta_h) \mathrm{d}t \tag{23}$$

where

$m_1$ proportional gain.
$m_2$ derivative gain.
$m_3$ integral gain.

The same analysis as the previous case can be done, considering an additional parameter $m_3$ as the output of the neural network. In this case, the neural network would have three outputs corresponding to the three parameters of the proportional integral derivative controller. The partial derivative of the torque with respect to this third parameter would be calculated easily taking into account the Eq. (23).

Although the previous analysis has been carried out considering the controller of one link, similar equations can be obtained for other link controllers.

## 3. Implementation

In the proposed method, the neural network outputs are the parameters of the controllers. Therefore, it can be said that the neural networks adjust automatically the parameters of the controllers. A PC Pentium computer has been used to implement the neural network controllers. In addition, A/D converters, D/A converters and optical encoders have been used in order to apply the torques to the robot joints and to capture the joint angles and joint velocities. A Cartesian trajectory should be specified along with an approach vector $\vec{a}$ and a sliding vector $\vec{s}$ for the gripper. As is well known the inverse kinematics will give a set of angles for each link.

In order to see the performance of the method, the PUMA robot was trained to follow an oscillating input. Several trials were carried out. In Fig. 4, the results of one trial are shown. The angles of the first joint along with the desired first joint angles are presented. In this case proportional derivative controllers have been used. The outputs of the neural networks have been multiplied by 10 in order to get greater values for the parameters and the weights have been initialised between $-0.1$ and 0.1 in order to avoid saturation in the neurons. As it can be seen the results are

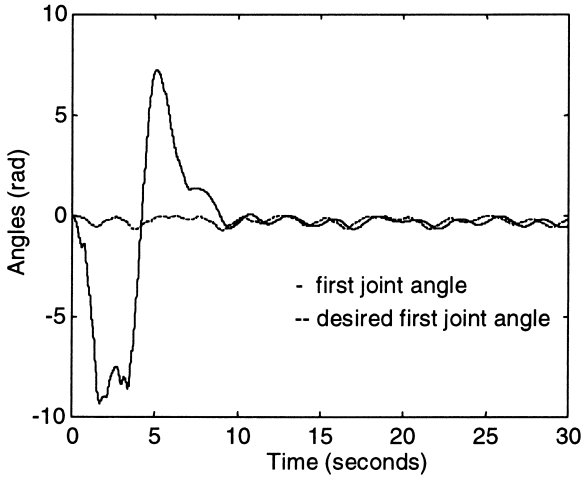


Fig. 5. Joint angles vs. time: second try.

not satisfactory; there is a big offset between the desired first joint angles and the actual first joint angles. However, the shape of the first joint angle curve is similar in some regions.

At this point, it seems convenient to test new kinds of controllers. In Fig. 5, the results corresponding to apply proportional integral derivative controllers are shown. That is, each neural network adjusts the three parameters corresponding to this kind of controller. The outputs of the neural networks have also been multiplied by a scale factor as in the previous case and the initialisation has been done in the same way. However a new factor has been introduced in order to keep the integral gain in an appropriate range of values. In addition to the multiplication by 10 of all the outputs, in this case the integral gains have been multiplied by 0.1, such that the integral gains given by the networks were small. Better results are obtained for this case. The big offset presented in Fig. 4 is eliminated. It can be appreciated that after some seconds the neural networks are able to adjust the parameters of the controller in order to follow the desired angles. As it can be seen in Fig. 5, one of the difficulties in the implementation is the robot response in the first interval of time, where the joint angles take arbitrary values and the robot carries out erratic movements. This problem comes from the kind of values chosen for the initial weights. The initial weights have been chosen as random numbers, hence the parameters are far away from the desired ones. In Fig. 6, the convergence of the parameters toward their final values is shown. This problem can be overcome by choosing convenient values for the initial weights rather than random values. One solution is to take as initial weights the ones obtained in a previous experiment with the same desired joint angles or similar ones.

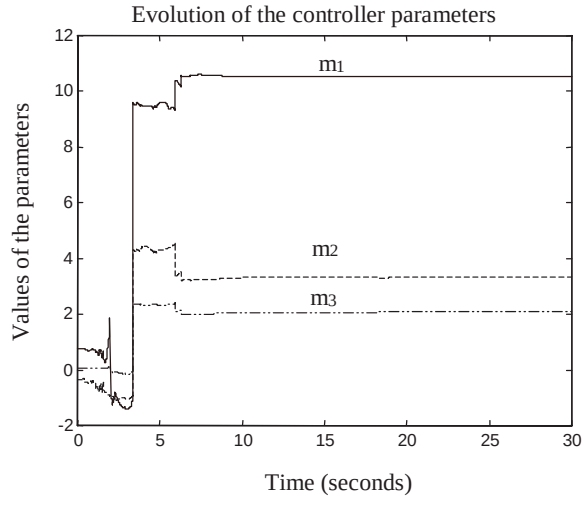


Fig. 6. Evolution of PID controller parameters.

## 4. Conclusions

In this paper, an approach based on neural networks has been presented to control a PUMA robot. Neural networks perform well in treating this kind of highly nonlinear systems by providing a good method for adjusting the parameters of the controllers. Concretely, proportional derivative controllers and proportional integral derivative controllers have been used for obtaining satisfactory results when the integral term is introduced.

It is important to remark that if new controllers were considered, the proposed training algorithm could be still applied, only it would be necessary to define the relation between the torque and the parameters of the controller. This characteristic makes this control approach very convenient in testing other controllers.

## References

[1] Spong MW, Vidyasagard M. Robot dynamics and control. Wiley, 1988.
[2] Pham DT, Liu X. Identification of linear and nonlinear dynamic systems using recurrent neural networks. Artificial Intelligence in Engineering 1993;8(1):67–76.
[3] Acosta L, Marichal GN, Hamilton A, Méndez JA, Moreno L. Design of a dynamic neural network structure for system identification, Intelligent Industrial Automation (IIA'96), 1996. pp. A77–A81.
[4] Moreno L, Acosta L, Marichal GN, Hamilton A, Méndez JA. A neural network structure for dynamic modelling of PUMA 560 Robot, Intelligent Industrial Automation, IIA'97, 1997. pp. 63–6.
[5] Miller III W, Sutton R, Werbos P. Neural networks for control. Cambridge: MIT Press, 1990.
[6] Narendra KS, Mukhopadhyay S. Adaptive control of nonlinear multivariable systems using neural networks. Neural Networks 1994;7(5):737–52.
[7] Kawato M, Furukawa K, Suzuki R. Hierarchical neural-network model for control and learning of voluntary movement. Biological Cybernetics 1987;57:161–85.
[8] Karakasoglu A, Sundareshan MK. Recurrent neural network-based adaptive variable structure model following control of multijointed robotic manipulators. Automatica 1995;31(10):1495–507.
[9] Karakasoglu A, Sudharsanan SI, Sundareshan MK. Identification and decentralized adaptive control using dynamical neural networks with application to robotic manipulator. IEEE Transactions on Neural Networks 1993;4(6):919–30.

[10] Acosta L, Méndez JA, Torres S, Moreno L, Marichal GN, Hamilton A. On the design and implementation of a neuromorphic self-tuning controller. Neural Processing Letters 1999, in press.

[11] Hush DR, Horne BG. Progress in supervised neural networks. IEEE Signal Processing Magazine 1993;1:8–39.

[12] Gupta HM. Fuzzy logic and intelligent systems. Dordrecht: Kluwer Academic, 1995.