Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 2, pp. 148-157

Applying RBF Neural Nets for Position Control of an Inter/Scara Robot

Fernando Passold

Pontifical Catholic University of Valparaíso
College of Electrical Engineering
Avenida Brasil 2147, Valparaíso, Chile
E-mail: fernando.passold@ ucv.cl

Abstract: This paper describes experimental results applying artificial neural networks to perform the position control of a real scara manipulator robot. The general control strategy consists of a neural controller that operates in parallel with a conventional controller based on the feedback error learning architecture. The main advantage of this architecture is that it does not require any modification of the previous conventional controller algorithm. MLP and RBF neural networks trained on-line have been used, without requiring any previous knowledge about the system to be controlled. These approach has performed very successfully, with better results obtained with the RBF networks when compared to PID and sliding mode positional controllers.
Keywords: manipulator robots, position-force control, neural networks.

1 Introduction

This paper describes and discusses practical results obtained with the use of computational intelligence techniques, specifically artificial neural networks, applied to the position control of a real manipulator robot. The neural controller proposed in this work was applied in a real scara robot installed at the Industrial Automation Laboratory of the Federal University of Santa Catarina, Brazil. The work is the result of a cooperation between the Automation and Mechanical Engineering departments. The robot was manufactured by the Institute of Robotics (IfR) of the ETH (Swiss Federal Institute of Technology, *http://www.ifr.mavt.ethz.ch/*). Differently from most industrial manipulator robots, this one has an open architecture, which allows the implementation of any type of control law. The main purpose was to evaluate new algorithms for position/force control, since the robot is also equipped with a force sensor. Fig. 1 shows the Inter/Scara robot.

Manipulator robots are a type of non-linear and time variant systems. Conventional controllers, such as PD and PID, are used among other advanced and robust controllers that require some knowledge about the dynamic model of the system under control. In the case of manipulator robots, it is difficult to obtain some parameters, as the inertia matrix and mass centers of any joint, with sufficient accuracy. Therefore, either adaptive controller are required to overcome these inaccurate parameters or control laws based on Lyapunov functions are developed to guarantee some kind of stability [1]. Both approaches require some knowledge about the system. This work explores a computational intelligence technique, namely artificial neural networks, to deal with this situation. In particular, control algorithms based on neural networks and fuzzy logic techniques are considered intelligent control approaches that do not require any previous knowledge about the system to be controlled. The main goal of this work is to explore an effective computational intelligence technique to control this complex system using a structure as simple as possible, with preference to one that requires less change in the previous conventional controller that has installed in the system, do not overload the main processor and is robustness against disturbance and load effect variations.

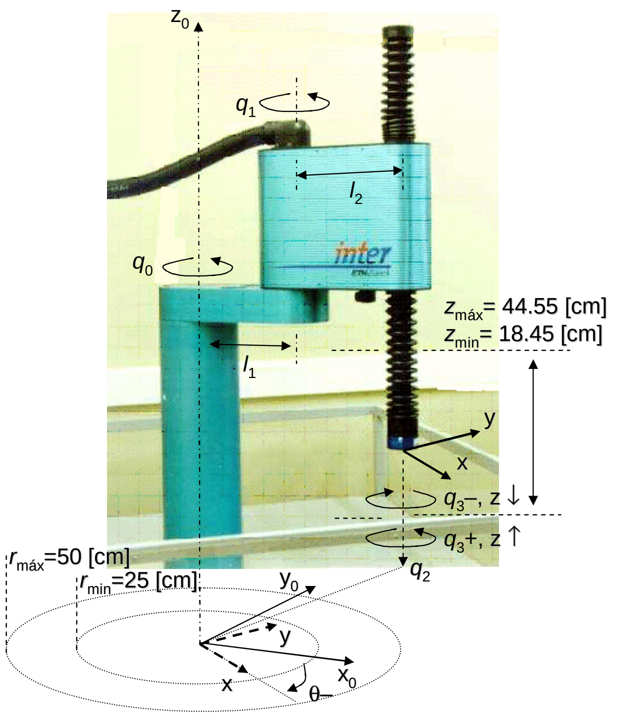


Figure 1: Inter/Scara robot and its coordinate system.

2 Neural Controller Types

Artificial Neural Networks (NNs) have been applied to several cases of control systems, showing special adequacy when we have to deal with complexity, non-linearity or uncertainty [13]. The neural approach is interesting, notably in the cases where:

a) Mathematical models of the process are poor or do not even exist and linear models are inadequate to represent the system with sufficient accuracy;

b) The system works satisfactorily with an existing controller but its performance deteriorates substantially when high performances (e.g. speeds) are required, so non-linear controllers become necessary.

NNs have proved their ability to approximate arbitrary nonlinear maps and the availability of methods for adjusting their parameters on basis of input and output data makes them particularly attractive in identifiers and controllers [2]. Narendra also comments that it seems to be valuable to keep linear and non-linear components working in parallel, where the neural networks represent the non-linear components [2]. He also mentions the brief learning time periods and the increase of accuracy that result from this kind of integration. In the control systems area, a few neural models have been proved to be more suitable than others, mainly:

1) Multilayer Perceptron networks (MLP), and;

2) Radial Base Function networks (RBF).

Among several ways to apply NNs in a control scheme, we can cite: (i) inverse identification (requires undesirable off-line training), (ii) reference model structure, (iii) internal model control, (iv) predictive control (uses two NNs, one of them trained off-line to identify the system), (v) optimal control (also requires two nets: the first one is used to quantify the state space of the system and the next acts as a classifier). Katič and Vukobratovič [3] and Morris and Khemaissia [4] discuss two learning architectures that seem to be the most appropriate and promising: a) Feedback-error learning architecture; and b) Adaptive learning architecture. The feedback error learning approach is characterized by the NN inserted in the feedback path to capture the nonlinear characteristics of the system. The ANN weights are tuned on-line with no off-line learning phase and, when compared with the adaptive technique, we do not require any knowledge of the robot dynamics, linearity in the unknown system parameters or the tedious computation of a regression matrix. Thus, this approach is model-free and represents an improvement over adaptive techniques [5].

2.1 Multilayer Perceptron Net

This network consists of a set of sensory input units (source nodes) that constitute the input layer, one or more hidden layers and an output layer. The input signal propagates through the network in a forward direction, on a layer-by-layer basis [6]. Multilayer perceptrons could be trained in a supervised manner using the back-propagation algorithm. Back-propagation is based on the error-correction learning rule and uses a least-mean-square error algorithm to adjust its connection weights. The error back-propagation learning consists of two stages: first, a forward phase, when the input vector is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, an output set is produced as the current response of the network. During the forward phase the weights of the network are kept unchanged. In the second phase, the backward phase, an error signal is propagated backward through the network against the direction of synaptic connections and the weights are adjusted to make the current response of the network move closer to the desired response based on a steepest descent algorithm, or back propagation weight update rule, also called generalized delta-rule [6].

2.2 Radial Base Functions Net

The basic structure of the RBF network consists of three layers. Different from the MLP networks, the layers here have different tasks. The first layer is passive and only connects the model to the real world. The second layer is a unique hidden layer. It performs a non-linear transformation from the input vector space to the internal vector space. The last layer is the output layer and transforms internal vector space into output vector space in a linear manner. There are several algorithms available to train the network [6, 7, 8]. These two types of neural nets can be universal function approximators [6, 8].

3 Controller Proposed

The controller proposed here uses a conventional PD or PID that performs in parallel with an artificial neural network trained on-line. This kind of architecture for neural controllers is known as feedback error learning because the net uses the output signal generated by the conventional controller as its own error signal that is back propagated for learning purposes ([2, 5]). Fig. 2 shows the architecture applied in this case.

Both types of NNs tested deal with the same input data vector: $\mathsf{x}_{\mathsf{NN}} = \begin{bmatrix} \mathsf{q} & \dot{\mathsf{q}} & \ddot{\mathsf{q}}_{\mathsf{d}} \end{bmatrix}$, where q is the vector of current joint positions; $\dot{\mathsf{q}}$ is the vector of current joint speeds (obtained through numerical differentiation); $\ddot{\mathsf{q}}_{\mathsf{d}}$ corresponds to the desired joint accelerations (like in other manipulators, there is no accelerometer available for each joint, so the desired acceleration computed by the path generator was used). These inputs were bounded into its maximum and minimum possible operational values for this robot and then scaled between the neural input range -0.9 up to +0.9 only for the MLP nets. Note that, different from MLP nets, the RBF nets do not need a scale procedure sice that they could deal directly with rough data, but it was necessary to organize the input data into three different classes: 1) joint positions, 2) joint speeds and 3) joint accelerations – see fig. 3. The idea behind organizing the data

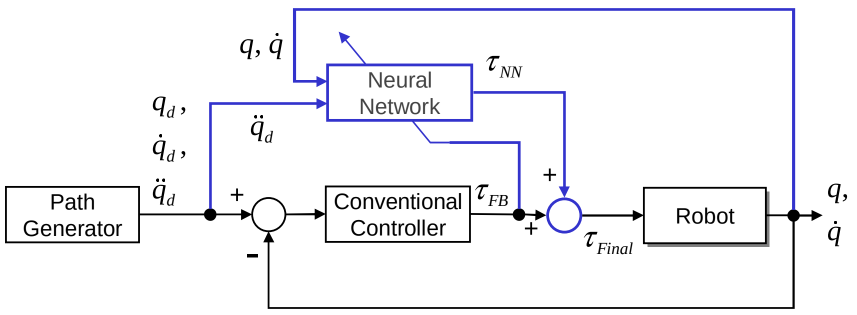


Figure 2: Feedback-error-learning neural controller used.

in different classes comes from previous skills with fuzzy inference systems. RBF networks could be compared with fuzzy inference systems [9]. Each class of input data could be understood as linguistic operations of fuzzy systems. Each class of data is mapped using m Gaussian functions and could be compared to the m membership functions that will be used in a fuzzy system. And finally, the rules and the way they are evaluated in a fuzzy system were performed by the output layer of a RBF network. Each synaptic connection of the output layer could be compared to the fuzzy IF-THEN-rules. The overall outputs are derived from the weighted sum done by the output layer [10]. Hence, m Gaussian functions were created to categorize each vector of each class of input data, as can be seen in fig. 3. It could be argumented that the massive amount of data required for the input layer (3 classes $\times$ 4 d.o.f. $\times\, m$ Gaussian functions) is a drawback of this approach, but this solution was related to the final application in mind in this case. Motions in the plane XY were done by the first two joints of the robot. Height (Z) and final orientation (θ) is performed by the last two joints of this robot but there is a mechanical coupling between them (a ball-screw-spline system), i.e., changes only in the final orientation of the robot result in a small change in the final height reached by the robot. That, represents an extra challenge to develop an effective controller to this kind of robot.

The centers x_i of the m desired Gaussian functions are fixed, based on the range (minimal and maximal values) of the input vector. That, allows defining the maximum Euclidian distance, d_{max}, between each Gaussian centers as: $d_{\max} = (x_{\max} - x_{\min})/(m-1)$ and then fixing the standard deviation (or spread) of each Gaussian function to be used according to:

$$\sigma = \frac{d_{\max}}{\sqrt{2m}} \tag{1}$$

The traditional back-propagation algorithm expanded with momentum term was used to adjust in real-time the weights of the MLP and RBF networks [6]. The addition of the momentum term to the delta rule traditionally used to update the weights of the net (based on the method of the descending gradient of the error signal), speeds up this algorithm, it eliminates decurrent oscillations of the calculation of the gradient and prevents the net to get paralyzed into a point of minimum local (and not global) in its surface errors [6]. Both networks end with 4 neurons, each one to evaluate the torque needed to command each joint motor of the robot. The final torque sent to each joint is defined as:

$$\tau_{Final} = \tau_{FB} + \tau_{NN} \tag{2}$$

where τ_{FB} is related to the torque evaluated by the conventional feedback controller that performs in parallel with one of these networks. PD and PID controllers working in the joint space have been used. The equation for the PID used is given by:

$$\tau_{FB} = \hat{B}(q)\left[K_p\tilde{q} + K_i\int \tilde{q}\,dt + K_d\dot{\tilde{q}}\right] \tag{3}$$

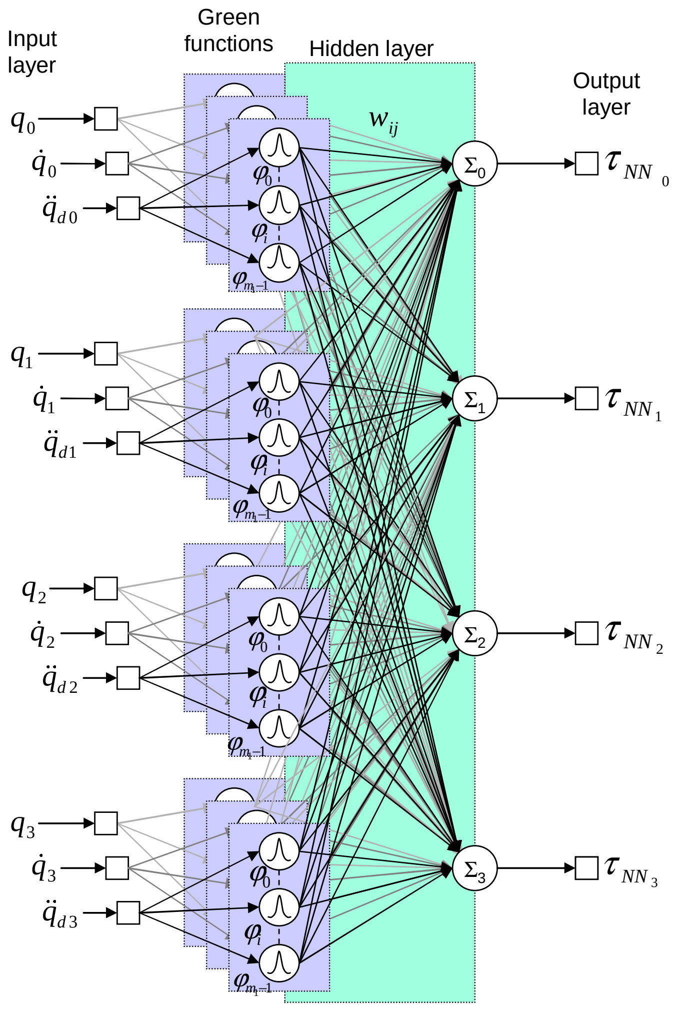


Figure 3: Structure of the RBF net implemented.

where $\hat{B}(q)$ refers to the inertia matrix of the robot (estimated); $\tilde{q} = q_d - q$ represents the error between the desired and the actual joint position; $\dot{\tilde{q}}$ refers to the velocity error; K_p is the vector of proportional gains for each joint; K_i refers to the integral vector gains and K_d is the derivative gain vector for each joint. To get a PD action over the system, the K_i vector was not used (equal to zero).

4 Experimental Results

The proposed controllers were implemented to the Inter/Scara robot (fig 1). The first two links of the Inter/Scara robot acts like an XY planar robot, and each one has 0.5 meters of length and its mechanical transmissions use harmonic drives (HD) to reduce motor-joints frictions at a minimal level. The last two joints use a ball-screw-spline mechanical scheme that allows movement in Z direction and the definition of the final orientation of these robot (angle θ). This robot could be manipulated in the Z direction between 18.45 (cm) and 44.55 (cm). Tab. 1 shows the Denavit-Hartenberg parameters among others of this robot.

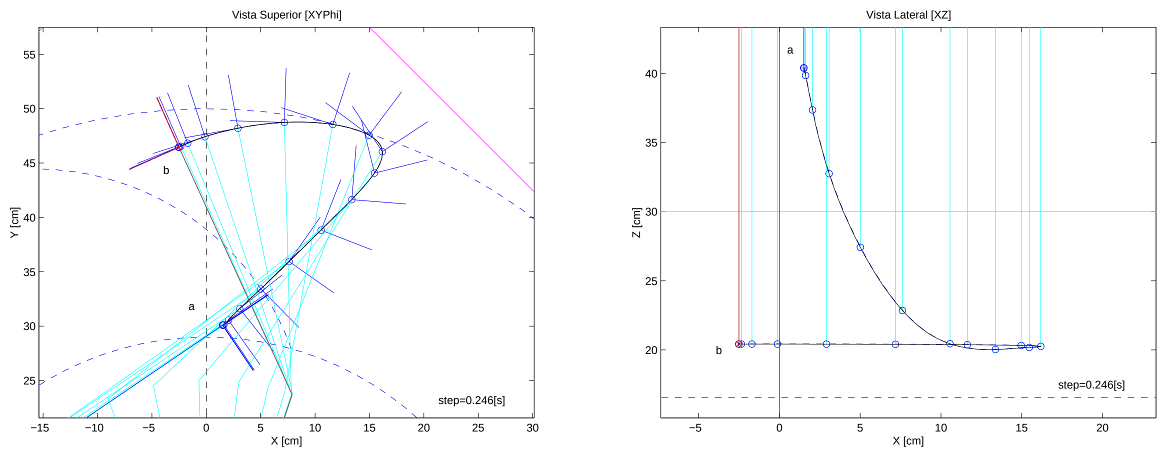
The Inter/Scara uses the XO/2 real-time object oriented operational system (*www.XO2.org*) developed by the Institute of Informatics of the Swiss Federal Institute of Technology (ETH). It includes a high level object oriented programming language called Oberon which is a kind of successor of Modula 2 and Pascal (see *http://www.oberon.ethz.ch*. The robot runs a XO/2 version over a PowerPC 200 MHz

Table 1: Denavit-Hartenberg parameters of Inter/Scara robot.

Joint	α_i	a_i	d_i	q_i	m_{l_i}	τ_{max_i}
0	0	250	665	q_0	$\cong 6.3$	333.0
1	180°	250	0	q_1	$\cong 19.5$	157.0
2	0	0	q_2	0	*	877.0
3	0	0	0	q_3	*	16.7
	(degrees)	(mm)	(mm)		(Kg)	(Nm)

Note: m_{l_1} includes the last 2 joints(*).

equipped with 16 Mbytes of memory, which communicates with its I/O devices using an industrial VME bus (67 MHz). The user interface is done through a TCP/IP connection with a PC running the Oberon System 3 for Windows Win32 2.3 (http://www.oberon.ethz.ch/) over the Windows 2000 Pro. The user develops the whole control system of the robot (including text command interface with the user, initialization and security functions) and through a cross compiler the execution programm is downloaded to the CPU of this robot (up to 4Mbytes of code). The algorithms for the proposed neural controllers were implemented as a real-time task running within a sampling rate of 1 millisecond. In each millisecond is evaluated the action of conventional controller, also it is evaluated the forward phase of the neural net and still the backward phase when the trajectory error, $\tilde{q} > 0.0001$(rad|m). The controller was tested over the trajectory shown by fig. 4. Table 2 shows the joint positions, speeds and accelerations developed for each joint. All the four joints were moved simultaneously.



(a) XYθ view. (b) XZ view

Figure 4: Trajectory used for the tests.

The PD/PID controller were digitally implemented using a 1(ms) sampling rate, and also an anti wind-up scheme was introduced with it to limit the integrative values up to 10000. Table 3 shows the parameters used for the tested PD/PID controllers. Their gains was settled initially based on Ziegler-Nichols parameters and then better adjusted using trial-and-error method.

Fig. 5 shows the output torques developed by different controllers for the joint 3 (the last and faster). Note the different performances developed by the PD+MLP1c (MLP with one single hidden layer), PD+MLP2c (MLP with two hidden layers), PD+RBF(5) (RBF with 5 Gaussian functions) and PID+RBF controllers tested.

Table 2: Parameters of the trajectory used for the tests.

Parameter	Joint 0	Joint 1	Joint 2	Joint 3
q_a	2.45	-1.85	-0.25	-1.57
q_b	1.25	0.75	-0.46	0.00
	[rad]	[rad]	[m]	[rad]
$\dot{q}_{máx}$	-0.71	1.19	-0.22	0.85
	[rad/s]	[rad/s]	[m/s]	[rad/s]
$\ddot{q}_{máx}$	0.85	1.00	0.47	-0.93
	[rad/s^2]	[rad/s^2]	[m/s^2]	[rad/s^2]

Table 3: PD/PID gains used.

	Joint 0	Joint 1	Joint 2	Joint 3
K_p	4900	12100	90000	14400
K_d	140	220	600	240
K_i	478	1200	9200	1410

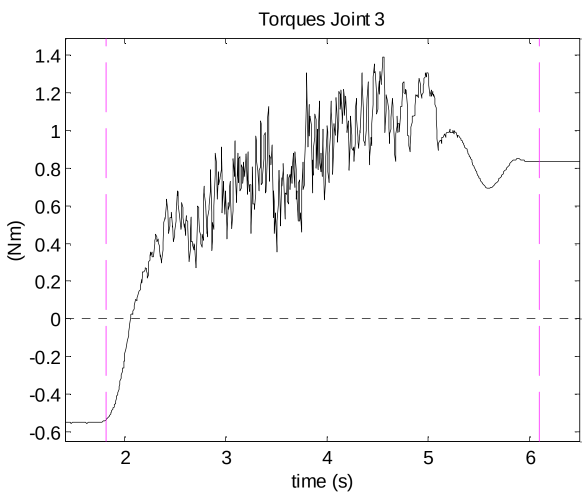
Fig. 6 shows the trajectory error during all the time. During the initial one-third of the robot configuration change period the NNs are in their learning time and the conventional controller still predominates in the joint control. But even before the end of this period of time, it could be seen that the NNs output torque takes predominance over the final torque evaluated (fig. 6). It could be seen that the NN learns the dynamic behavior of the system and then does the dynamic compensation that results in higher performance compared to a conventional controller.

The best results for the MLP nets were achieved with learning rate $\eta = 0.035$ and momentum term $\alpha = 0.5$. Related to the RBF nets, the best learning parameters founded were: $\eta = 0.005$ and $\alpha = 0.5$. Note that a PID performing with a RBF net, allows the better performance (fig. 5(d) and 5(e)) followed by the PD+RBF, PD+MLP and finally, the single PD. It was noted that the use of two hidden layers for the MLP NN does not imply in a better performance and moreover adds a small residual memory effect in presence of a disturbance on the system (this behavior has been observed in tests where an elastic string was placed in the middle of a linear trajectory).

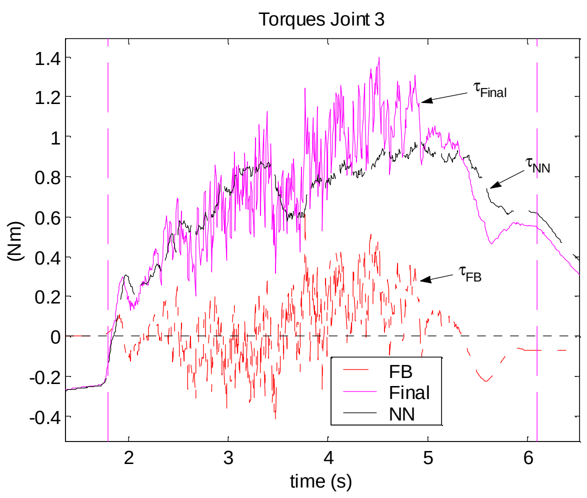
Experimental results have also demonstrated that the addition of more than 5 Gaussian functions in the RBF NN controller, slightly increase the performance, but at the expenses of a significant higher computational cost.

5 Conclusions

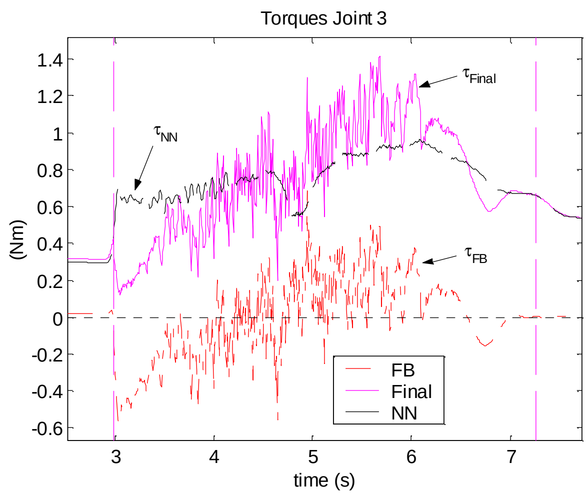
This paper has presented a practical and successful application of a neural controller performing in parallel with a conventional controller in the position and trajectory following control of a real robot. The use of a conventional controller performing in parallel with the NNs is advantageous to maintain the robustness of the system when the NN become saturated (due to high learning rates) and it is important to force the readjustment of the synaptic weights of the NN used when the robot changes its configuration. As soon as the NN captures the dynamic behavior of the system, the final torque is given quite totally by the NN and a higher performance could be achieved. Both the MLP and RBF ANN perform very well,



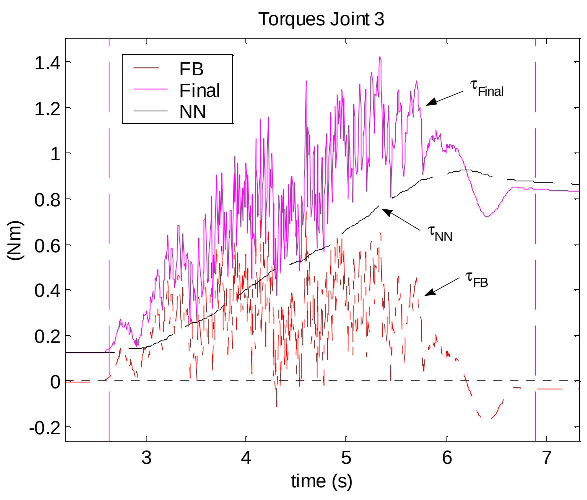
(a) PD controller output torques.



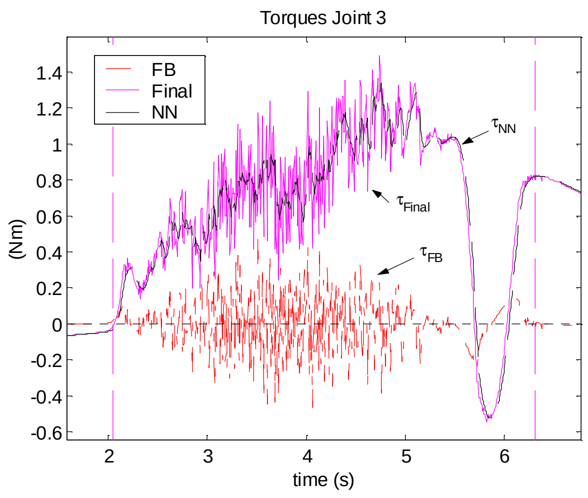
(b) PD + MLP1c controller outputs torques.



(c) PD + MLP2c controller outputs torques.



(d) d) PD + RBF(5) controller outputs torques.



(e) PID + RBF(5) controller outputs torques.

Figure 5: Controllers output torques.

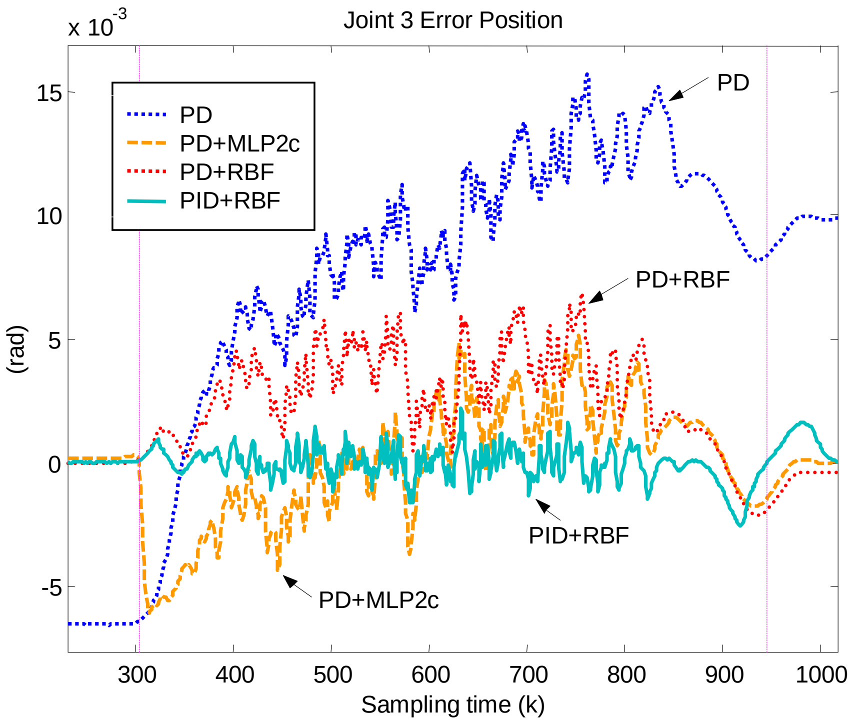


Figure 6: Joint 3 some tracking error.

but RBF does it better and faster than a MLP, mainly if it works in parallel with a PID controller. An additional and unexpected advantage could be achieved with the PID+RBF controller: robot motion with lowest noise levels (quite silent). On the other hand, there is also a drawback: NN requires more processing power to work in parallel with the conventional controller – see table 4 and fig 7 that summarize the results achieved by the different controllers.

Table 4: Power computer resources required.

	Processing time	
Controller	Min	Max
PD	104	104
PD + MLP2c	194	385
PD + RBF(5)	333	579
PD + RBF(7)	425	679
PD + RBF(9)	104	809

Note: values expressed in microseconds (µs).

Even if computer resources are short, a simple PD+MLP with one hidden layer allows better results than a PD controller. Otherwise, if higher processing power is available, a PID+RBF achieves the best results. Since the neural controllers proposed here have performed very successfully, future directions of this works intend to establish an integrated position/force control over a hybrid control architecture to deal with robot applications that imply some contact with the environment.

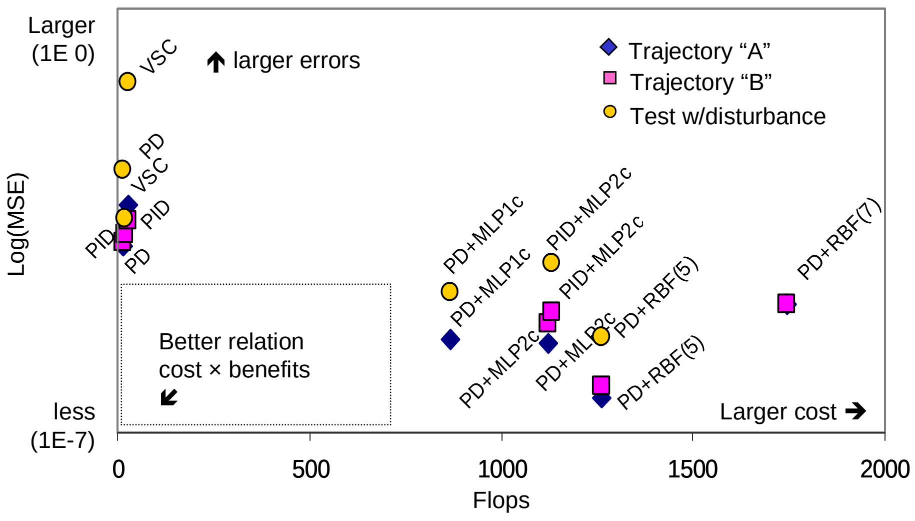


Figure 7: Relationship within computational cost × benefit of the controllers tested here.

Bibliography

[1] Sciavicco, L. and Siciliano, B., *Modeling and Control of Robot Manipulators*, McGraw-Hill, 1996.

[2] Narendra, K.S., Neural networks for real-time control. In *36th IEEE Conference on Decision and Control - CDC'97*, 1026–1031, San Diego, California, USA, 1997.

[3] Katič, D. and Vukobratovič, M., Connectionist based robot control: an overview. In *13th IFAC*, volume 1b-05 6, 169–174. San Francisco, USA, 1996.

[4] Morris, A.S. and Khemaissia, S., Artificial neural network based intelligent robot dynamic control. In A.M.S. Zalzala and A.S. Morris (eds.), *Neural Networks for Robotic Control – Theory and Applications*, chapter 2, 26–63, Ellis Horwood, Great Britain. 1996.

[5] Kim, Y.H. and Lewis, F.L., Neural network output feedback control of robot manipulators. *IEEE Transaction on Robotics and Automation*, 15(2), 301–309, 1999.

[6] Haykin, S., *Neural Networks A Comprehensive Foundation*, Prentice Hall, New Jersey, USA, 2nd edition, 1999.

[7] Gabrijel, I. and Dobnikar, A., Adaptative RBF neural network. In *SOCO'97 Conference*, 164–170. Nimes, France. URL `http://cherry.fer.uni-lj.si:80/~gabriel/soco97/soco97.zip`, 1997.

[8] Girosi, F. and Poggio, T., Networks and the best approximation property. In M.M. Gupta and D.H. Rao (eds.), *Neuro-Control Systems, Theory and Applications*, 257–264. IEEE Pres, Piscataway, New Jersey, USA, 1993.

[9] Fritzke, B., Incremental neuro-fuzzy systems. In *Applications of soft computing, SPIE International Symposium on Optical Science, Engineering and Instrumentation*, San Diego, 1997.

[10] Kiguchi, K. and Fukuda, T., Intelligent position/force controller for industrial robot manipulators – application of fuzzy neural networks, *IEEE Transactions on Industrial Electronics*, 44(6), 753–761, 1997.