Machine Intelligence & Robotic Control, Vol. 3, No. 1, 7–26 (2001)

Paper

A New Partitioned Robot Neurocontroller: General Analysis and Application to Teleoperator Modeling Uncertainties Compensation*

Spyros G. Tzafestas†, Platon A. Prokopiou†, and Costas S. Tzafestas‡

Abstract: This paper presents in a compact unified way some new results on manipulator dynamics identification and control using neural networks (NNs). First, the issue of robot identification is considered. The neural net is divided into three subnetworks, each one corresponding to a part of the robot dynamics. Aiming at enabling online adaptation to changes in the dynamic model without spoiling the role of these subnets, a novel Heuristic Error Distribution technique is introduced, theoretically analyzed, and supported by simulations. This incorporates prior knowledge about the robot dynamics not only in the network structure, but also in the training algorithm. Multilayer Perceptron (MLP) and Radial Basis Function (RBF) Networks are employed. A teleoperator architecture available in the literature is used to test the control performance of the proposed neural network structure. This architecture is enhanced to compensate for modeling uncertainties. Simulation results obtained with the neurocontrollers with 2 and 3 DOF manipulators, as well as with a sliding-mode controller, are reported. A final performance comparison slightly favors MLPNN.

Keywords: Manipulator Dynamics Identification, Neurocontroller, Teleoperation, Multilayer Perceptron, Neural Networks, Radial Basis Functions

1. Introduction

NEURAL Networks (NNs hereafter) have gained over the past years increasing attention from the engineering community and were applied to various fields, such as signal and image processing, control, robotics, forecasting and economics. Although justified criticism has been expressed by the classic control community on the novelty of the concepts that NNs introduce [4], their success should cause no surprise: NNs can approximate a wide range of nonlinear functions by a simple learning procedure, using input-output data efficiently, so that a system small in size and easily implemented, by both hardware and software, with parallel processing capability is generated. The field of robotics involves several challenging NN application areas. NNs can be employed at most parts of a robotic system (environment recognition, decision making, path planning, control, etc). Especially in large complex systems, such as legged robots [5], [6], cooperating manipulators [7] and telemanipulators [8]–[11], where the models at hand become very complicated, they can offer simple and valuable solutions.

Several applications of NNs on single robotic manipulators have been reported in the literature (e.g. [12]–[19]). In this paper, NNs are exploited at the control level and a new solution to the manipulator control task is presented. The whole controller is divided in three subnetworks, each one identifying a part of the robot dynamics. Such a modular NN architecture results in a simpler, faster and easier to train controller, offering good insight on the current system situation, but complicates the online training. To overcome this, a special Heuristic Error Distribution method (the HERD method) is designed and analyzed in detail, which reduces the confusion of the networks' role when large modeling errors arise, e.g. when the robot picks up an object or after an accident. With HERD we effectively incorporate prior knowledge about the specific control problem not only in the structure of the NN, as current consensus dictates, but also on the training algorithm. This method is our main contribution to the robot control problem.

The method can be employed to any robotic system, and can be extended to assist in the control with partitioned NNs of other classes of systems, as well. Here we discuss its application to a teleoperator system. Thus the effectiveness of the method is verified in a demanding system, where the interaction of two manipulators with the environment and the non-passive human operator can cause instability and performance degradation much easier than in a single manipulator. In addition, several issues of the application of NNs in teleoperator control are explored, indicating solutions to practical aspects arising at the implementation of the system. Extensive simulation results are presented throughout the paper to support the proposed approach.

Teleoperators will continue to be valuable in the future for space, underwater, underground, hazardous or medical applications. Even though research on autonomous robots is expected to reduce the need for humans working in hostile or unpredictable environments, our supreme recognition, analysis, decision-making and manipulation abilities, are not easy to be matched. Current research topics include the incorporation of autonomous functions [9], [20], and the design of schemes robust to time delays in the communication channel between the master and slave robots [21]–[23]. Virtual reality techniques are also gaining a lot of

* Received October 10, 2000; accepted January 6, 2001. This paper is a unified and upgraded version of three papers presented at the IEEE ICNN'97 [1], ISIC'97 [2], and ISIAC'98 [3] conferences.

† Intelligent, Robotics and Automation Laboratory, Signals, Control and Robotics Division, Department of Electrical and Computer Engineering, National Technical University of Athens, Zografou Campus, 15773, Athens, Greece. E-mail: tzafesta@softlab.ece.ntua.gr

‡ Institute of Informatics and Telecommunications N.C.S.R. "Demokritos," Aghia Paraskevi Attikis, Athens 15310, Greece.

Paper No. 1345-2681/01/010007-20

attention [24].

NNs, possessing a remarkable ability to identify and control strongly nonlinear and multivariable plants, with minimum need for prior knowledge, through an adaptive, compact and fast system, can provide valuable solutions at various levels of a telemanipulator. The methods developed for single manipulators can be applied locally to the master and slave. In addition, they can assist in the coordination between the human operator and the machine (e.g. visual representation, force feedback redefinition, incorporation of human and environmental dynamics in the control loop, etc). Certain autonomous functions can also be assigned to NNs. However, very few applications of NNs in this field are reported [1], [2], [25], [26].†

Our case study is based on the teleoperator architecture proposed by Lee and Lee [9], [22], [27], [28]. In the traditional concept of telepresence, according to which the exact position and force sensed at the slave side is fed back to the operator, so that he feels as if he is "physically present" at the slave workspace [29], is abandoned. Lee and Lee argue that in the context of semiautonomous control, this concept might mislead the operator, since he is not aware of the automatic functions but "feels" their results. Doubts about the necessity of telepresence have also been previously expressed [29]. An alternative goal for teleoperators is defined as trying to execute directly the actions determined by the operator's intention, rather than just duplicating his arm's movements [9]–[11], [30]. Other interesting contributions in teleoperation research include [31]–[37].

In this paper, their scheme is modified in order to be able to compensate modeling uncertainties, arising after the slave picks up an object of unknown mass and shape, or caused by hardware malfunctioning, accidental deformation or deliberate model simplification. In [38] a sliding robust controller was designed to ensure the achievement of the desired performance. Here, Multilayer Perceptron Neural Networks (MLPNNs) and Radial Basis Functions Neural Networks (RBFNNs) are utilized for the same purpose [1], [2].

The paper is organized as follows: In Section 2 the robot identification and control problems are discussed in detail, and the HERD method is introduced. Simulation results support the theoretical analysis. Section 3 is dedicated to the teleoperator system and the modifications we propose. Simulation results are reported, utilizing both MLP and RBF NNs. A short presentation of the performance obtained with sliding mode controllers is made, and the three types of controllers are compared. The conclusions are presented in Section 4.

†Lee [27] proposed a general architecture, termed "Sensing-Knowledge-Command Fusion Network," for performing "interactive and cooperative sensing and control," which in essence functioned in the object recognition and task planning level. Although this network was not referred to as "neural," parts of it could be realized with neural networks, fuzzy systems, expert systems or other techniques based on probability theory.

2. Robot Dynamics Identification and Control Using Partitioned Neural Networks: The Herd Method

2.1 A previous NN partitioning approaches

In a robot controller, NNs are most effective when employed for the identification of the robot's dynamics. The identified model can then be employed within a Model Reference Adaptive or a Computed Torque control architecture. For rigid-link manipulators of n degrees of freedom (DOFs), considered in our contribution, this dynamic model is described by (see e.g. [39], [40]):

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau_a - J^T(q) f_e \quad (1)$$

where q is the vector of joint variables, $D(q)$ the inertia matrix, $C(q,\dot{q})\dot{q}$ the vector of Coriolis and centrifugal forces, $g(q)$ is related to gravity, τ_a is the vector of driving forces, f_e represents the forces acting at the end effector from the environment, and $J(q)$ is the Jacobian matrix. This widely accepted representation of the manipulator dynamics, offers an inherent, "physical," division of the dynamics in three terms, relating to

a) the inertial (D),
b) the Coriolis, centrifugal and viscous (C) and
c) gravitational terms (g).

Therefore, one can partition the identifying NN to subnets, each one representing one of the above terms (**Fig. 1**). The forms of the nonlinear functions that these subnets have to learn are more or less a priori known and can be taught offline.

According to Lewis *et al.* [14] the partitioning of an NN to neural subnets:

1) simplifies the design,
2) gives added controller structure, and
3) makes for faster weight tuning algorithms.

Our simulations confirmed that significantly less weights are required for representing, within a given error margin, the robot dynamics with partitioning to subnets, than with employing one big NN. It is not the aim of this paper to further elaborate on the advantages of partitioning, as this is considered adequately covered by the literature, but to provide an efficient training algorithm for this type of manipulator neurocontrol.

Several researchers up to date have proposed partitioned neural controllers to identify the robot dynamics. In the method of Guez and Selinsky [12], each subnet is trained offline during appropriate movements of the robot that isolate and identify the terms. First the gravitational terms are being taught through the execution of position commands, so that the other terms of the robot dynamics become zero. Then, by imposing constant velocity the weights of the Subnet C are identified and, at the end, the Subnet D is being trained. During the online operation of the NN as a controller, the subnets' weights remain fixed. Only the output layer weights, i.e. the relative contribution of each nonlinear function and/or subnet are being constantly adapted. A similar partition to subnets was employed by Kawato *et al.* [13]. Lee *et al.* [41] proposed the division to only two NNs: a fixed, offline trained MLPNN, and a smaller RBFNN, trained online to provide adaptation. Ziauddin

and Zalzala [19] utilized one NN for each joint, not as a standalone controller, but in order to assist a classic controller in compensating for unmodeled dynamics, such as friction and payload variations.

A different approach was proposed by Lewis *et al.* [14]. The basic nonlinear functions are not identified but explicitly incorporated in the NN: the network structure was carefully determined in order to match the a priori known robotic model of Eq. (1). They employed two layer subnets. The weights of the first layer were fixed, so that linearity in the parameters for the NN holds, and thus some theorems guaranteeing convergence could be proven. This first layer could be chosen to represent some general basis functions. The most suitable ones for a manipulator controller are the sine and cosine functions as well as their products with each other and with the joint position and velocity. Since Eq. (1) is explicitly known, it is easy to determine the basis functions needed for any type of robot. The overall control scheme realizes an effort to extend the classic adaptive control theory to NN control theory.

In the present contribution a different approach is followed. In the method of [14] explicit knowledge of the robot dynamics is assumed. In the Guez and Selinsky approach [12], if one chooses as basic functions each of the terms in Eq. (1), then any change in the robot parameters would lead to a new offline training phase. If one chooses the sine and cosine functions multiplied by the joint velocity and acceleration, then a lot of subnets are required (7 NNs plus 26 trainable weights for the final layer for a 2-DOF revolute-link robot). The second choice as well as the method of Lewis *et al.* [14] leave no room for the NN self-optimization through learning and the identification of new terms, arising e.g. upon contact with an object or at malfunctioning.

We searched for a method that would:

(a) enable on-field adaptation to sudden parameters' change,

(b) enable fast output calculation and weight tuning (for this small NNs with high parallelism are necessary),

(c) preserve the basic nonlinear functions learned offline (an inappropriate online training method could for example allow Subnet G to learn some parts of the $\boldsymbol{D}$ matrix),

(d) allow the learning of new nonlinear terms, induced due to contact with the environment, actuator saturation or hardware damage, and

(e) encompass minimum a priori information in the subnet *structure*, so that the net self-optimization and a compact representation are possible.

In order to meet requirements (b) and (e) the NN was divided to only three subnets (**Fig. 1**) corresponding to matrices $\boldsymbol{D}, \boldsymbol{C}$ and $\boldsymbol{g}$ of Eq. (1). Their outputs were added at a final layer. The only a priori information used was the subnet inputs. In order to achieve online adaptation, the subnets were chosen to have no fixed part. To fulfill point (c) above, a novel, heuristic way to train the subnets was developed. The final algorithm utilizes both measurable data (actually $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$) as well as information based on the structure of the robot. We named it HERD (Heuristic ERror Distribution) method.

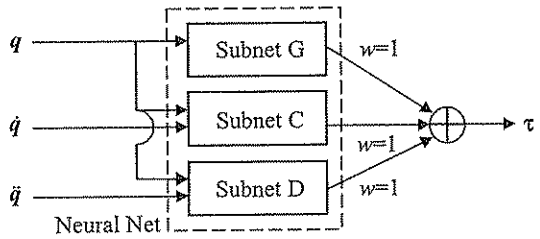

Fig. 1 Partition to subnets

Our initial investigations on the topic were conducted with MLPNNs [1], [38]. Our initial results utilizing RBFNNs, are reported in [2]. This Section summarizes and presents in a unified way all our efforts in the problem of identifying the robot dynamics, under the presence of uncertainties, with partitioned NNs, making use of minimal information.

In the sequel we will use the following notation: τ: overall torque, e: overall error. τ_D, τ_C, τ_G, e_D, e_C, e_G: torque/error related to each subnet. In our setup (Fig. 1) $\tau = \tau_D + \tau_C + \tau_G$, $e = e_D + e_C + e_G$ and $\boldsymbol{q} = [q_1 \;\; q_2 \cdots q_n]^T$ (generally, the subscript denotes vector index).

2.2 Altering the training algorithm instead of the network topology

The dynamics of an MLPNN can be represented as:

$$y = W^T \sigma(V^T x) \tag{2}$$

where $\boldsymbol{V}, \boldsymbol{W}$ are weight matrices for the hidden and output layers respectively, $\sigma(\cdot)$ is the NN nonlinear function, and $\boldsymbol{x}, \boldsymbol{y}$ the vectors of inputs and outputs respectively. By considering the three subnets we yield:

$$\begin{aligned}\tau_G + \tau_H + \tau_D = \boldsymbol{W}_G^T \sigma(\boldsymbol{V}_G^T \boldsymbol{x}_G) + \boldsymbol{W}_H^T \sigma(\boldsymbol{V}_H^T \boldsymbol{x}_H) \\ + \boldsymbol{W}_D^T \sigma(\boldsymbol{V}_D^T \boldsymbol{x}_D)\end{aligned} \tag{3}$$

As is well known, the Back Propagation (BP) algorithm is in fact the application of the gradient descent rule to the specific topology of the MLP. Applying the gradient descent rule to the partitioned controller of Eq. (3), we yield:

$$\begin{aligned} & w_{s,ij}(k+1) \\ & = w_{s,ij}(k) - \eta \frac{\partial E(\boldsymbol{W})}{\partial w_{s,ij}} \\ & = w_{s,ij}(k) - \eta \frac{\partial}{\partial w_{s,ij}} \left[\frac{1}{2} (y_d - y)^2 \right] \\ & = w_{s,ij}(k) - \eta (y_d - y) \frac{\partial y}{\partial w_{s,ij}} \\ & = \begin{cases} w_{s,ij}(k) - \eta (y_d - y) \dfrac{\partial \tau_G}{\partial w_{G,ij}} & \text{if } s = G \\ w_{s,ij}(k) - \eta (y_d - y) \dfrac{\partial \tau_H}{\partial w_{H,ij}} & \text{if } s = H \\ w_{s,ij}(k) - \eta (y_d - y) \dfrac{\partial \tau_D}{\partial w_{D,ij}} & \text{if } s = D \end{cases} \end{aligned} \tag{4}$$

where $w_{s,ij}$ denotes the weight between the i cell (in layer k) and the j cell (in layer $k+1$) of subnet s ($s = G, H$ or D), $E(w) = 0.5(y_d - y)^2$ the error function, η the learning coefficient. The splitting of the derivative to three

independent parts is due to the fact that each subnet's output depends solely on its own weights. The remaining partial differentiations, which adhere to the NNs of each submodel, can be computed through the BP equations. Notice that (4), i.e. the application of the gradient descent rule to (3), is equivalent to applying the BP to each subnet, with the *whole* robot dynamics identification error issued at each one of them.

BP is a *general* training (i.e. parameter estimation [4]) algorithm, applying to the *specific* NN topology and a *wide* class of nonlinear functions to be approximated. As Sjöberg *et al.* [4] pointed out, the most efficient nonlinear optimization methods are based on iterative local search in a "downhill" direction from the current point, of the kind:

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} - \mu_i \boldsymbol{R}_i^{-1} \nabla \hat{f}_i \tag{5}$$

where $\hat{\theta}^{(i)}$ is the parameter estimate after iteration number i, μ_i the step size, $\hat{f}_i$ the identified model, $\nabla \hat{f}_i$ an estimate of the gradient, and $\boldsymbol{R}_i$ a matrix that modifies the search direction. The gradient descent corresponds to $\boldsymbol{R}_i = \boldsymbol{I}$, whereas other choices lead to algorithms performing a search along the Gauss-Newton and Levenberg-Marquadt direction. Although such algorithms are widely accepted and of proven efficiency, they remain *general*, "*black box*" ones, in that they incorporate no knowledge of the plant to be identified. They only involve derivatives of the specific *model components (regressors)*, i.e. in the MLPNN case the mapping $\hat{f} = \boldsymbol{W}^T \sigma(\boldsymbol{V}^T \boldsymbol{x})$.

It is the aim of this paper to demonstrate that enhanced performance can be obtained if the peculiarities of the robot dynamics are utilized, in a "gray-box" identification scheme: adopting a loose notation, by suitable $\boldsymbol{R}$ in (5), the search direction is biased towards a "manipulator specific" direction. Broadly thinking, the idea of tuning the training algorithm of a general approximator according to hints about the plant, rather than modifying the approximator's topology, could also be beneficial for other types of plants.

In the case of RBF NNs, the dynamics are represented by:

$$\tau_a = \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x})$$

where $\boldsymbol{W}$ is a weight matrix and $\boldsymbol{\phi}$ a nonlinear "radial basis" function. The derivation of a training algorithm for the subnet output level is similar to the above and, for the case of gradient descent, yields exactly the same result. Any other type of NN or generally trainable nonlinear mapping which adheres to (5) (wavelets, B-spline networks, etc), could in principle benefit from a similar subnet implementation.

In the next section, the specification of search direction biasing functions, as well as trade-off between relying on little previous knowledge of the robot dynamics, against accurately distributing the error to each subnet, is explored.

2.3 Error distribution to subnets: problem analysis

In an offline training phase and if a nominal model is known, the training of each subnet is easy and can be accomplished by any standard learning algorithm. In the online case, when modeling uncertainties are present, distributing the total error among subnets becomes an almost "blind guess." It is then very difficult to preserve the role of each subnet.

In our effort to avoid utilizing a detailed mathematical model, the following information is everything we consider known:

(a) The total error $\boldsymbol{e}$ per joint,
(b) The joint and end-effector (Cartesian) position, velocity and acceleration,
(c) A nominal dynamic model in a mathematical form (Eq. (1)) or input-output experimental data from the manipulator, used to train the NN offline, to determine its size and inputs, and to specify the type of norms (used e.g. in HERD),
(d) The robot dynamics in the general form of Eq. (1). We basically exploited the fact that $\tau_D(\tau_C)$ is proportional to $\ddot{q}$ ($\dot{q}$) and independent of $\dot{q}$ ($\ddot{q}$), and τ_G is independent of both $\dot{q}$ and $\ddot{q}$,
(e) Hints about the present dynamic model (e.g. the fact that the output τ_C for a 2-DOF revolute link robot is proportional to $\sin(q_2)$), and
(f) May be some bounds of the uncertainty/torque.

On the other hand, the following are considered unknown:

(a) The error of each subnet $\boldsymbol{e}_D, \boldsymbol{e}_C, \boldsymbol{e}_G$ (magnitude and sign)
(b) The present dynamic model parameters.

After outlining the problem as above, let's investigate the possible effects of modeling uncertainty. We can distinguish two situations:

(i) an object of unknown dynamic parameters is lifted and carried or manipulated, so that, assuming that the object is firmly grasped, the parameters of the last link (only) change;
(ii) due to an accident, more links are deformed and thus the structure of the dynamic model changes (even the DOFs can change if for example a link is cut off).

The second case is extremely severe, since the only useful information left is $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ and $\boldsymbol{e}$. In our work only the most realistic case (i) was considered. For case (i) the mathematical functions involved remain unmodified, i.e. the uncertainty is parametric (although these parameters are "hidden" in complicated nonlinear expressions).

In the sequel we have to decide whether: A) to consider the mathematical model for the robot (Eq. (1)) as known, or B) not to use it. A mathematical model can be used either: A1) explicitly, to derive control laws, or A2) indirectly to derive "hints." If the mathematical model is to be used, then an analytical adaptive control law might be more convenient, especially for 2 or 3 DOF robots, or robots that are decoupled in two 3-DOF subsystems. Since the power of NNs lies in the fact that they do not require such detailed prior information, the actual choice is between (A2) and (B).[†] Whether we choose options (A2) or (B), the decisive difference with (A1) is that we actually have in our disposal for processing pairs of input-output data approximated by the NN in a input/output function but not the internal dynamics that cause it. That is, a function $y = f(x)$ is identified, but (in contrast to the mathematical model) we don't know if e.g.: $f(x) = 1$ or $f(x) = \sin^2(x) + l\cos^2(x)$

[†]Using the classical identification terminology, cases (A1), (A2) and (B) correspond to a "white," "gray" and "black" box identification task [4].

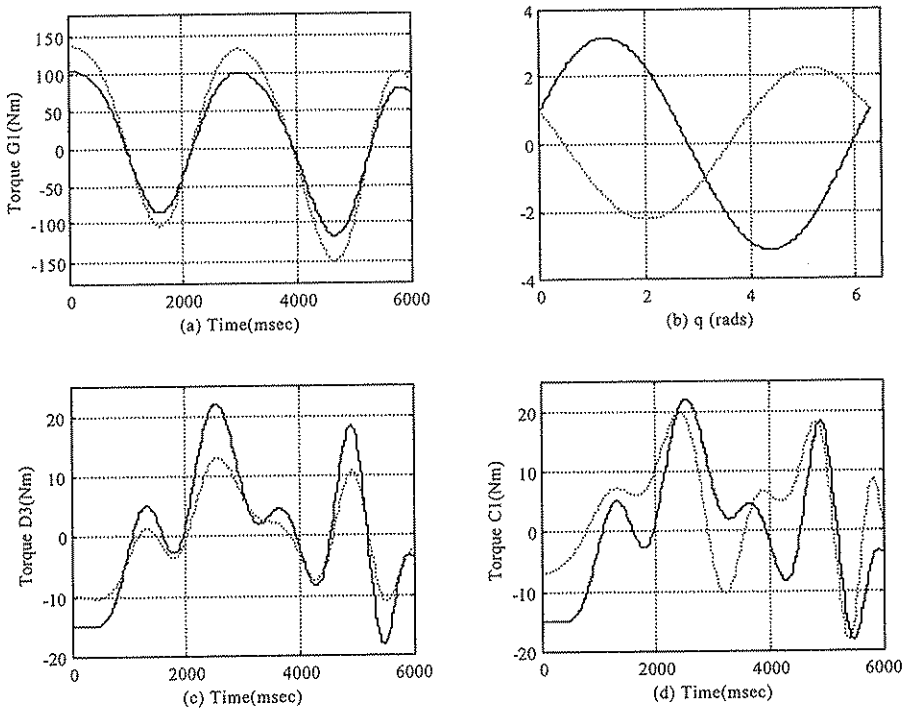

Fig. 2 (a) τ_{G1} for 2-DOF robot, modified with $m_2 = 7.5$ and $l_2 = 1.5$, which is the most complex case in a partitioned 2-DOF robot controller; (b) plot of the functions for nominal ($3 \sin q + \cos q$) and modified ($-2 \sin q + \cos q$) models; (c) τ_{D3} for 3-DOF robot, modified with $m_3 = 1.5$ and $l_3 = 0.36$, which is one of the most nonlinear (strange-shaped) torques for the 3-DOF robot; and (d) τ_{C1} for 3-DOF robot, modified with $m_3 = 1.5$, $m_2 = 0.25$, $l_2 = 0.35$, and $l_3 = 0.36$, where note that a change in l_2 and m_2 is not realistic (only possible in a variable structure robot), the solid line denotes the nominal model, and the dashed line is for the modified model

with link length $l = 1$. In the second case a change in l will make the trigonometric terms reappear, whereas in the first not. What we want to stress here is that we are not allowed to predefine the influence of each parameter's (e.g. l's) modification to the dynamics.

In order to reduce the problem into easier subtasks, we can observe that each of the robot torque's element is determined by a complicated matrix ($\boldsymbol{D}, \boldsymbol{C}$ or $\boldsymbol{g}$) specified by the structure of the robot, which is multiplied by a fixed measurable vector ($\dot{q}, \ddot{q}$, or 1). The effect of the *measurable* part is fixed for all robots and easily detectable, and thus can be readily exploited in an error distributing algorithm. On the contrary, the influence of the *structural* part is subject to parameter modifications and needs to be further analyzed. In the preliminary analysis that was provided at the beginning of this section, this type of data covers types of "known information" (c) and (e). The algorithms presented in the sequence exploit separately each of these multiplicative parts of the terms of Eq. (1).

Let's try to assess whether the structural information is of any use in our task. We will argue that *the nominal dynamic model can serve as a fair first approximation to the altered manipulator dynamics*, since in the majority of cases they have the same general form.

We considered it helpful to categorize the changes in the robot dynamics due to uncertainties, *according to their effect on the sub-torques of Eq. (1)*, in:

(I) Changes in robot type (structural changes), where the model/DOFs change a lot.

(II) Usual parameter modifications, causing the type of the input/output function of the sub-torques to change. For example $f(q) = (m+l) \sin q + \cos q$, can be modified from $3 \sin q + \cos q$ to $-2 \sin q + \cos q$. This totally changes the shape of the resulting total function (see **Fig. 2**b).

(III) Usual parameter modifications, but not causing the type of the input/output function to change. For example $f(q) = (m+l) \sin q$ will always be a sine, crossing zero and getting its maximum value at fixed positions although the magnitude is variable. Such parameters are multiplicative or additive to nonlinear functions, but the coefficients of a weighted sum of them are not so (see Fig. 2a).

Note that the distinction to types (II) and (III) is valid regardless of the linearity in the parameters, which holds in both the examples mentioned above. Notice also that it *only* takes into account the input/output function, and not the underlying nonlinear functions, which remain unmodified. To the best of the authors' knowledge, the literature up to date only distinguishes between structural uncertainty (type I) and parametric uncertainty (types II+III).

In the case of robot dynamics, certain important constraints are imposed: a) Their parameters are positive. b) Not any change mathematically possible is likely to happen. It is realistic to assume that serious uncertainties in a robot can only arise in the last link. c) The uncertainty

Table 1 Manipulators used in simulations testing the error distribution algorithms

2-DOF ROBOT: Revolute joints	
link 1:	$m_1 = 10, l_1 = 1, l_{c1} = 0.5, I_1 = 0.8$
link 2:	$m_2 = 5, l_2 = 1, l_{c2} = 0.5, I_2 = 0.45$
3-DOF ROBOT: Revolute joints	
link 1:	$m_1 = 10, l_1 = 1, l_{c1} = 1$
link 2:	$m_2 = 5, l_2 = 0.7, l_{c2} = 0.7$
link 3:	$m_2 = 1, l_2 = 0.3, l_{c2} = 0.3$
Modeling Errors	
Unless otherwise stated, 50% on the mass and moment of inertia of the last link	
Dynamic Equations	
Well known for the above, see e.g. [39]	
TASK	
2-DOF robot:	$q_1 = \pi + 0.8\pi\cos(2\pi \times 0.2kT)$),
	$q_2 = \pi - 0.8\pi\cos(2\pi \times 0.16kT)$.
3-DOF robot:	$q_3 = \pi + 0.8\pi\cos(2\pi \times 0.22kT)$;
	$q_2 = \pi + 0.8\pi\cos(2\pi \times 0.2kT)$;
	$q_1 = \pi - 0.8\pi\cos(2\pi \times 0.16kT)$.
The movements cover a representative part of the joint space and demand joint speed (acceleration) up to 4 [rad/sec] (4 [rad/sec^2]).	

is, realistically thinking, bounded (Note, however, that no bounds are assumed in the algorithms of Section 2.4).

Due to the above reasons, when the controller is partitioned as in Fig. 1, the change in the form of the robot dynamic functions, is hardly ever as serious as for the example in (II) above. The changes due to a realistic parameters' modification are mostly of type (III) above or not significant (i.e. of type "II degenerated to III"). This helpful simplification is due to the partitioning in three subnets as in Fig. 1. Thus, the *nominal model* is a fair coarse approximation. Based on the old model one can guess when the subnet output (and hopefully its error too) will be 'big,' 'small' or 'zero,' 'increasing' or 'decreasing,' etc. Figures 2a, c, d illustrate these arguments. Refer to **Table 1** for task and manipulator parameters used throughout this paper.

An examination of the dynamics of various robots reveals that the above conclusion is reasonable for most of the cases. For other types of robots with up to 3-DOFs (e.g. spherical or cylindrical) [39] the equations are as simple as for the 2- or 3-DOF revolute link robot, whereas for 6-DOF manipulators the dynamics are often decomposed in two 3-DOF subsystems [42], where the required change in torque is like the one in Fig. 2c. For sub-torques that remain of type (II), another way to specify the error can be used (e.g. a fixed percentage of e) or the HERD method (described in the next paragraph) can be utilized as a first-order approximation. One can decide which part is of type (II) or (III) by either observing the nominal dynamics or experimenting offline.

2.4 Error distribution algorithms: formulation

Five different ways to distribute the error were examined (**Table 2**) namely:

Table 2 Error distribution methods

Method	Error Distribution Coefficient
Simplistic	1/3
Coarse	$\lvert\tau_i\rvert/\lvert\tau\rvert$ with $i = D, C, G$
Measurable Info (MI)	f_{mi}
HERD	$(\lvert\tau_i\rvert/\lvert\tau\rvert) f_{mi}$ with $i = D, C, G$
(Error per Subnet) = (Error Distribution Coefficient) ×(Total Error)	

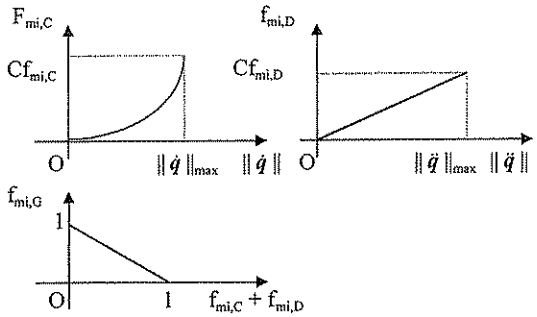

Fig. 3 Error distribution functions f_{mi}

a) Back Propagation of the total error;
b) an equal distribution of 1/3 of the total error to each subnet ("simplistic" solution);
c) a distribution relative to the contribution of the subnet to the total output of the net, ("coarse" solution);
d) a distribution exploiting the information about the "measurable part" ("Measurable Information (MI) solution"); and
e) a combination of the two previous methods, named HERD Method (Heuristic ERror Distribution Method).

Since, as shown in Section 2.2, the plain BP requires that each subnet is trained with the total error e, the error distribution methods examined are mathematically formulated as functions multiplying e, or as implementations of $\boldsymbol{R}$ in (5). Equivalently, they can be thought of as modifications of the learning coefficient $\eta(t)$, which has now become time-dependent. Thus, the methods proposed here effectively constitute variations of BP, and generally gradient-descent methods, specifically designed for manipulator control. Our methods are not significantly more ad hoc than gradient descent itself. The identification convergence and the stability of the whole control system can most likely be proven within specific control architectures (as e.g. in [14], [15]), and remain for future work.

The simplistic solution coincides with the plain BP, with a lower learning coefficient. This fact was not stressed in our previous papers [1]–[3], [26]. The results obtained with it are qualitatively equivalent to the BP, although in some simulations performance differences may arise due to the very fact that η is altered.

The coarse method is based on the assumption analyzed in Section 2.3, namely on the assumption that the general form of the dynamic model will not change for realistic parametric modifications.

The MI method was introduced in [1] (although this name was not used). Some modifications were however

made. The form of f_{mi} was inspired by observing the way in which each term of the robot dynamics depends on $q, \dot{q}$, and $\ddot{q}$. These dependencies are plotted in **Fig. 3** and described by:

$$\begin{aligned} f_{mi,D}(\|\ddot{q}\|) &= Cf_{mi,D}\|\ddot{q}\| \\ f_{mi,C}(\|\dot{q}\|) &= Cf_{mi,C}\|\dot{q}\|^2 \\ f_{mi,G}(\|\ddot{q}\| + \|\dot{q}\|^2) &= 1 - f_{mi,C} - f_{mi,D}. \end{aligned} \tag{6}$$

The specification of the coefficients Cf_{mi} is not an easy task. No fixed coefficients, globally optimal for any robot and any working conditions, can be found. Instead, a procedure to specify them is recommended: first the maximum realistic values of $\dot{q}$ and $\ddot{q}$ are specified. For this value, the mean of the subnet output divided by the total controller torque is calculated, over as many points as possible in the joint space. If the calculation can be performed on-line, the mean could be computed over the area of interest only. The nominal dynamic model is used, or, a model "close" to the new dynamics. However, the effect of the magnitude of the maximum allowable $\dot{q}$ and $\ddot{q}$ was much more important than the use of the nominal or another model. Thus the "point" $(\dot{q}, \ddot{q}) = (\max(\dot{q}), \max(\ddot{q}))$ is a critical one, determining the maximum efficient area of the algorithm. The Subnet C was the most sensitive part to the initial offline learning inadequacies. †

Note that each subnet output is assigned a different f_{mi}, i.e. an f_{mi} with different coefficients. The type of the norms in f_{mi} can also be adjusted according to the task and the robot. A reasonable choice is to make them dependent only on the subnet inputs (e.g. for the 2-DOF robot, the norm of subnet C's second output should be independent of $\dot{q}_2$). An additional heuristic used is that if $\|\ddot{q}\| < \epsilon$ (small constant) (or $\|\dot{q}\| < \epsilon$ (small constant)), the subnet's D (or C) training error is set equal to the subnet output. Thus the subnet output is forced to zero and a small deadzone is created around $\ddot{q} = 0 (\dot{q} = 0)$, aiming at healing a possible inadequate offline learning of the measurable part in this region.

The HERD method is clearly a concatenation of the coarse and MI methods. In essence, the various training sections of Guez and Selinsky [12] are here overlapped and take simultaneously place, in a fuzzy logic-like way.

Among the problems that could not be solved, two are the most important. First, although a guess on the magnitude of the error is made, its sign remains unknown. It was observed in our simulations, that when no training is made, most of the time the subnet error has the same sign as the subnet output. However, this information must not be exploited during the online adaptation, since it was found to lead to instability the coarse method, and thus also the HERD method. Second, the *total* error is minimized very soon (less than 0.3% in less than 20 sampling periods). After this period the learning is very slow, since the error to be distributed is small. If in the first few sampling periods the networks confuse their roles, then this confusion will not heal much in the sequence. This was unfortunately the case in the 2-DOF simulations reported in Section 2.5. For example, a total error of 0.5 [Nm] was divided to, say, 0.3 [Nm], 0.2 and 0.1, but were actually reflecting errors of, say, 15 [Nm], -10 [Nm], -4.5 [Nm]. In mathematical notation, $e = e_D + e_C + e_G$, but $|e| \ll |e_D| + |e_C| + |e_G|$. Several solutions were examined without obtaining significant improvement. Note that in the above example the sum of subnet errors is *not equal* to the total error: the distribution algorithm was allowed to selectively magnify the error, since the total error is much smaller than the real ones per subnet.

As a consequence of the efforts up to now, we conclude that an online-trained multipartitioned network can, under the presence of severe uncertainties, *act successfully as a controller*, since the total error is kept very small (0.1%–0.5%) but *cannot always avoid confusing the role of its subnets*.

Table 3 Neural network parameters

RBF Networks, 2-DOF robot			
Subnet G:	2–49–2 neurons,	inputs:	$q, \eta = 0.1$
Subnet C:	3–81–2 neurons,	inputs:	$q_2, \dot{q}, \eta = 0.01$
Subnet D:	3–81–2 neurons,	inputs:	$q_2, \ddot{q}, \eta = 0.01$
MLP Networks, 2-DOF robot			
Subnet G:	2–9–2 neurons,	inputs:	$q, \eta = 0.1, \alpha = 0.2$
Subnet C:	5–9–2 neurons,	inputs:	$q, \dot{q}^2, \dot{q}_1 \times \dot{q}_2, \eta = 0.01$ $\alpha = 0.2$
Subnet D:	4–10–2 neurons,	inputs:	$q, \ddot{q}, \eta = 0.01, \alpha = 0.2$
MLP Networks, 3-DOF robot			
For all subnets: $\eta = 0.000001, \alpha = 0$			
Subnet G:	3–15–3 neurons,	inputs:	q
Subnet C:	6–15–3 neurons,	inputs:	$q, \dot{q}$
Subnet D:	6–15–3 neurons,	inputs:	$q, \ddot{q}$

2.5 Simulation results

Initial simulations were performed with the 2-DOF manipulator of Table 1. Modeling uncertainty was 50% on the mass and moment of inertia of the 2nd link. The NN was initially trained at the nominal model, and then tried to learn "online" the modified dynamics.

Three of the methods presented in Section 3 were compared. A reasonable critical point for the determination of the coefficients of the HERD method was chosen to be $(\dot{q}, \ddot{q}) = (5[\text{m/sec}], 5[\text{m/sec}^2])$. This results in quite high velocities for industrial robots. The coefficients were calculated on the basis of the nominal dynamic model. The final values used were:

$$\begin{aligned} &Cf_{mi,G1} = 0.27, \quad Cf_{mi,G2} = 0.22, \quad Cf_{mi,C1} = 0.55 \\ &Cf_{mi,C2} = 0.6, \quad Cf_{mi,D1} = Cf_{mi,D2} = 0.1. \end{aligned}$$

To validate the methods, in conjunction with the mean error, a special index, named the *Confusion Index CI*, was used, in an effort to isolate the error caused by the error distribution algorithm from the normally appearing error, which is due to the learning algorithm, the specific NN structure, the initial offline learning quality, and the cho-

† In principle, Subnet C could be abolished, by calculating τ_C through: $h(q, \dot{q}) = \text{col}\left[\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial D_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial D_{ij}}{\partial q_k}\right)\dot{q}_j\dot{q}_k\right]$. However, since D is represented by an NN, it is not possible to isolate the D_{ij}s.

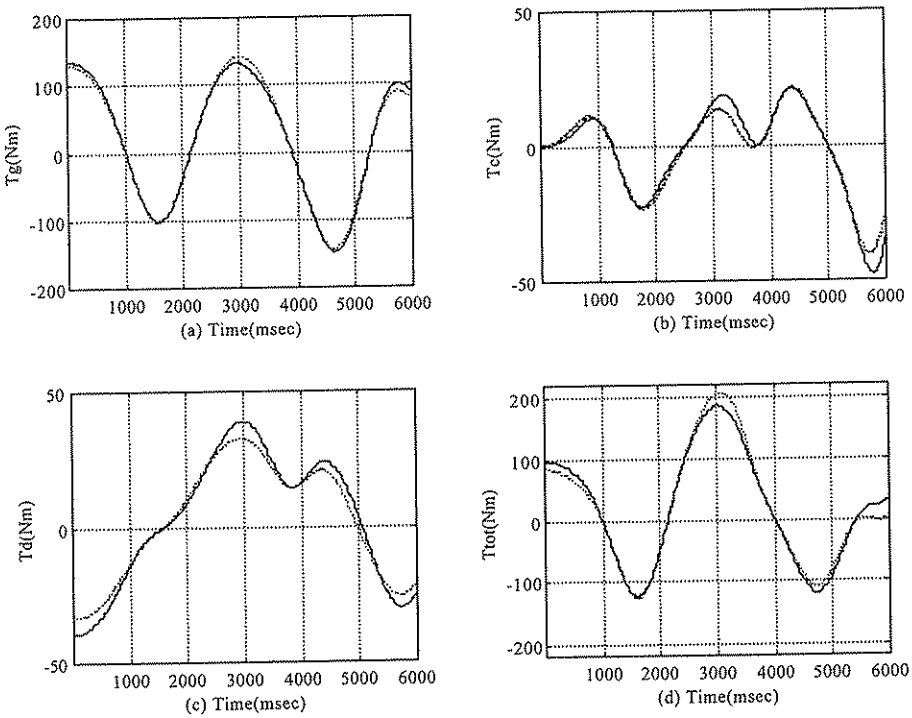

Fig. 4 2-DOF case with RBFNNs, representing the desired output and the output with HERD by the solid and dashed lines for the first joint: (a) Subnet G; (b) Subnet C; (c) Subnet D; and (d) the whole network

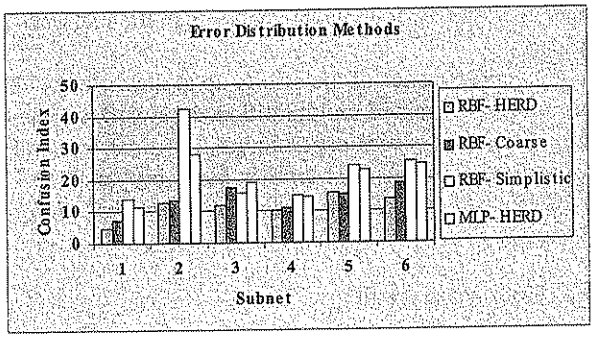

Fig. 5 2-DOF case with error distribution methods: representative results, where $1 = \tau_{G1}, 2 = \tau_{G2}, 3 = \tau_{C1}, 4 = \tau_{C2}, 5 = \tau_{D1}$, and $6 = \tau_{D2}$

sen parameters' values:

$$CI = |(\textit{subnet error}) - (\textit{subnet error by training with real error})|/(\textit{absolute subnet output} + 1) \times 100\%$$

In the denominator above, 1 is added so that the percentage does not become infimum when the desired output is zero.

Both MLPNN and RBFNN were tried. Their parameters are depicted in **Table 3**. Each of the RBF subnets was first trained offline with the standard Least Mean Square method [43], to learn the corresponding term of a nominal model. In this way the required basic functions were formed. The subnets were trained online using a gradient descent procedure outlined in [43]. The offline training data was forming a sparse grid over the input space. Neither the positions nor the spreads of the centers (i.e. the input norm) were trained, since the initial simulations indicated satisfactory performance without such a computation-costly training. These parameters were selected before learning. MLPNNs were trained with the same procedure, using, at all times, the standard BP algorithm with a momentum term and flat-spot elimination [43].

Some representative results are shown in **Figs. 4, 5**. Note that the trial trajectory is a hard one. From these results, the superiority of the HERD algorithm over the other methods is evident. Although the error per subnet is significant, it is acceptable since a poor qualitative information is only used.

The results obtained using RBFNNs were significantly better, as far as the subnet confusion is concerned. However, a direct comparison is not possible (Table 3). The MLPs had an extra input, $\dot{q}_1$, which they learned to ignore, but the algorithm used for RBF learning did not involve input norm modifications, and thus the net inputs had to be exact. The size of the RBF net was also much bigger than that of MLPs, and the resulting control torque with RBFs had significant chattering (refer to Section 3.4.3). On the other hand the initial offline training in RBFs is much simpler.

Simulations also revealed that for the algorithm to be satisfactory, the *offline training has to be good.* The zero-crossing points of the structural part are of special importance. The zeroing points due to the measurable parts are not so dangerous, since a special heuristic was incorporated to account for them (Section 2.4.3). In our tests $\epsilon = 0.75(0.0)$ for subnet C (D). It was also made clear, that for the above formulation of the error, the *contribution of*

Table 4 MLP 3-DOF error distributing methods

Subnet	BP (NME)	Coarse (NME)	HERD (NME)	BP (CI)	Coarse (CI)	HERD (CI)
$\tau_{total,1}$	0.0093	0.0032	0.0033	0.0611	0.0183	0.0126
$\tau_{total,2}$	0.0429	0.0392	0.0381	0.1890	0.0602	0.0283
$\tau_{total,3}$	0.0150	0.0039	0.0026	0.0652	0.0225	0.0433
$\tau_{G,1}$	0.0004	0.0003	0.0002	0.0358	0.0254	0.0239
$\tau_{G,2}$	0.0535	0.0404	0.0380	0.2694	0.2036	0.1892
$\tau_{G,3}$	0.0124	0.0047	0.0042	0.3446	0.1300	0.1182
$\tau_{C,1}$	0.0092	0.0034	0.0032	0.2242	0.0805	0.0720
$\tau_{C,2}$	0.0057	0.0033	0.0031	0.1305	0.0844	0.0760
$\tau_{C,3}$	0.0024	0.0005	0.0012	0.0923	0.0215	0.0534
$\tau_{D,1}$	0.0015	0.0005	0.0004	0.0611	0.0183	0.0126
$\tau_{D,2}$	0.0151	0.0050	0.0024	0.1890	0.0602	0.0283
$\tau_{D,3}$	0.0012	0.0004	0.0009	0.0652	0.0225	0.0433

NME = (Mean absolute error) – (Mean absolute error, when training with the real error per subnet)

CI: Confusion Index. (NME equals the CI numerator)

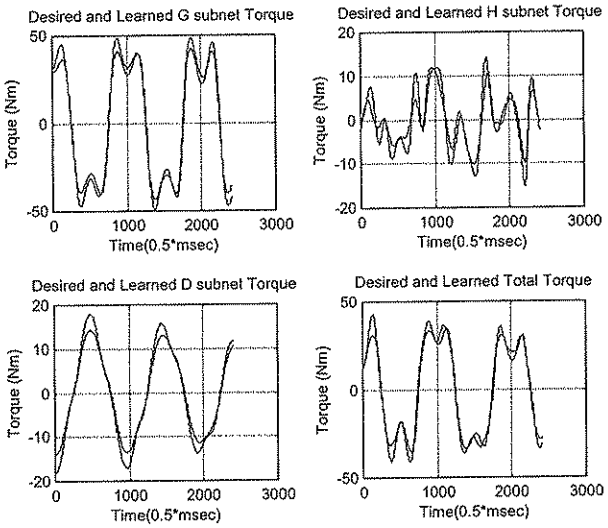

Fig. 6 3-DOF case, representing the desired output and the output with HERD by the solid and dashed lines: each subnet and the whole network for the first joint

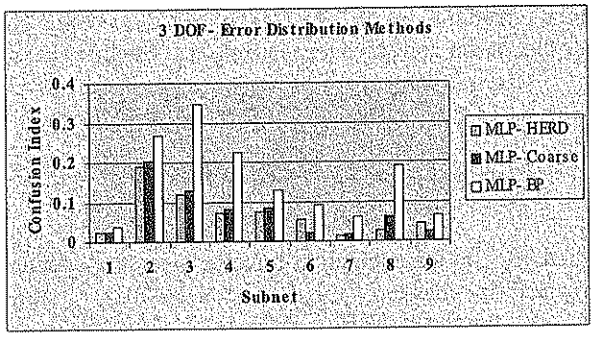

Fig. 7 3-DOF case with error distribution methods: representative results, where $1 = \tau_{G1}, 2 = \tau_{G2}, 3 = \tau_{G3}, 4 = \tau_{C1}, 5 = \tau_{C2}, 6 = \tau_{C3}, 7 = \tau_{D1}, 8 = \tau_{D2}, 9 = \tau_{D3}$, and notice that the error is much smaller than for the 2-DOF case

subnet C to the total error is critical. By giving the correct error to C, the error of the other two subnets decreases significantly. On the contrary, if we use the correct error only for G (or D), although the total error decreases, the error per subnet does not improve much. The error distribution to Subnet C is the weak point of the HERD method for the 2-DOF case.

Simulations were also performed *for the 3-DOF revolute-joint manipulator* of Table 1. The modeling error was 50% at the mass and 20% at the length of the last link. The NN was initially trained at the nominal model, and then tried to learn "online" the modified dynamics. A reasonable critical point for the determination of the coefficients of the HERD method was chosen to be $(\dot{q}, \ddot{q}) = (5[\text{m/sec}], 5[\text{m/sec}^2])$. The final values used were:

$$Cf_{mi,G1} = 0, \quad Cf_{mi,G2} = 0.4642$$
$$Cf_{mi,G3} = 0.3121, \quad Cf_{mi,C1} = 0.8013$$
$$Cf_{mi,C2} = 0.3174, \quad Cf_{mi,C3} = 0.5536$$
$$Cf_{mi,D1} = 0.1987, \quad Cf_{mi,D2} = 0.2185$$
$$Cf_{mi,D3} = 0.1344.$$

MLPNNs were tested for various learning parameters and trajectories. At the simulations reported here, the standard BP algorithm was employed, without a momentum term or other enhancement, since its performance was satisfactory.

Some representative results are shown in **Table 4** and in **Figs. 6, 7**. The three last methods of Table 2 were compared against the BP of the error to the subnets, instead of the qualitatively equivalent simplistic method of Fig. 5.

In contrast to the 2-DOF case, the error per subnet is remarkably small. This shows that the role confusion of each subnet depends on various factors (the learning algorithm, the specific NN structure, the initial offline learning quality, and the chosen parameters' values), However, in all cases the HERD algorithm is clearly superior to the other methods.

Notice also that, as expected, in all methods the biggest part of the actual error is due to the inherent inability of BP to train the subnets within one cycle, i.e. online: even if the real error is issued to each subnet, the error is significant, and about 2 levels of magnitude bigger than the one shown in Table 4. This error is of course small in relation to the NN output torque, as shown in Fig. 6.

3. Teleoperator Neurocontrol

3.1 An original control scheme

The teleoperator design concept of Lee and Lee [9], [22], [28] redefines force feedback, by actively altering the force fed to the operator and incorporating to it a term depending on the slave tracking error. Moreover, the control loop includes models of the human arm and the human reaction to force and vision stimuli during his effort to reach the desired position and force. The effects of time delay were also discussed in [9], [22], [28].

The overall control architecture of Lee and Lee is designed in the Cartesian space (**Fig. 8**). In order to provide compliant force control, a previous extension of Impedance Control proposed by the authors is utilized. The desired manipulator impedance is defined through a set of differ-

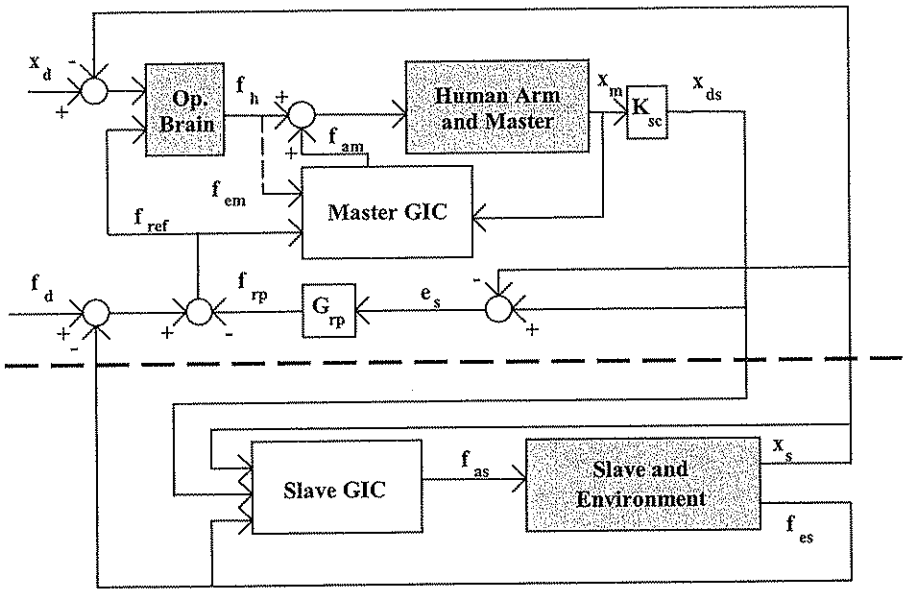

Fig. 8 Lee and Lee teleoperator system [9], [22], [28]

ential equations, termed the *Generalized Impedance* (GI). For the master and slave respectively it is specified as† :

$$f_{ref} + f_{em} = M_{dm}\ddot{x}_m + B_{dm}\dot{x}_m \tag{7}$$

$$M_{ds}\ddot{x}_s + B_{ds}(\dot{x}_s - \dot{x}_{ds}) + K_{ds}(x_s - x_{ds}) = B_{fs}\dot{f}_{es} + K_{fs}f_{es} \tag{8}$$

where f_{em} is the interaction force between the operator's arm and the master, f_{ref} is the reflected force, $x_{ds} = K_{sc}x_m$, the matrices K_{sc}, M_{ds}, B_{ds}, K_{ds}, B_{fs}, K_{fs}, M_{dm}, B_{dm} are parameters of the GI, and $f_{es} = Z_e(x_e - x_s)$ represents the contact force at the slave side, where Z_e is the environmental impedance and x_e its location. In order to impose the GI, a control law following the computed torque method was proposed.

Assuming ideal performance, the master and human arms form a dual system described by:

$$f_h + f_{ref} = (M_h + M_{dm})\ddot{x}_m + B_{dm}\dot{x}_m \tag{9}$$

where f_h is the intentional force of the operator, (i.e. the force applied by his brain to his muscle). Note that f_h is the response of his nervous system to the stimuli of vision and force (generated through a screen and the master arm). Equation (9) reveals that f_{ref} is the reaction force actually felt by the operator. In a conventional scheme it would be equal to (a scaled) f_{es}. Lee and Lee [9], [22], [28] define it as a combination of the force and position tracking error, and thus they actually redefine force feedback.

3.2 Dynamic model uncertainty in teleoperation

Modeling errors can be caused by several factors. On the slave side, the parameters of the last link change when a large object of unknown mass and shape is picked up. This is a common situation in telerobotic systems: one can easily imagine the manipulator collecting samples from an ocean floor or picking up a tool from an unstructured environment and performing an assembling task with it, or, while carrying it, pushing a door open. Another cause of errors is an accidental deformation of the end effector or even the last links. Finally, unidentified nonlinear terms (induced by an unexpected object or force field) or terms deliberately omitted in the basic design process to simplify the design, also fall in this category and need to be compensated. With the simulation tasks defined in Section 3.4 several of these situations are tested.

On the master side modeling errors are not likely to be large, since the master operates in a safe and well-known environment. Possible causes of uncertainty are now unidentified terms, deliberate simplification of the model and accidental deformation of the links. An appealing application for the master would be to provide the possibility to change its end effector, so as to best suit the task to execute. For example in a telesurgery system [8], [35] the surgeon would like to customize on spot the shape and feel of the handle he grips according to the instrument used at the slave side and personal preferences. An adaptive or robust controller can then be utilized to provide automatic adaptation. Uncertainty also becomes significant if the human arm dynamics, which are inherently nonlinear, time-varying, and difficult to analyze [10], [11], are considered. Even in this case, 20% seems a logical upper bound for master side modeling errors.

In [9], [20] the results of several simulations and experiments, with various control laws (other than GIC) for the slave are provided and discussed. The role of modeling errors was not considered in these papers. In [22] a new control scheme for applications with significant time delay is proposed. The new controller was reported to perform well also under 5%–10% errors in the slave model. However, this does not nullify the need for a new controller: some new terms added in the last version, which as a side effect compensate modeling uncertainties, decrease exponentially as the time delay decreases. At short time delays their contribution vanishes and the new version coincides

† In the above and in the following, subscripts m, s, h, and e will denote the master, slave, human and environment respectively.

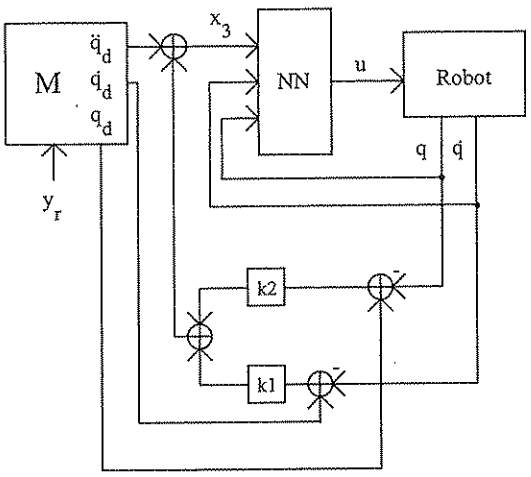

Fig. 9 Neurocontroller architecture employed in this paper (refer to [47], [48] for accurate error expression)

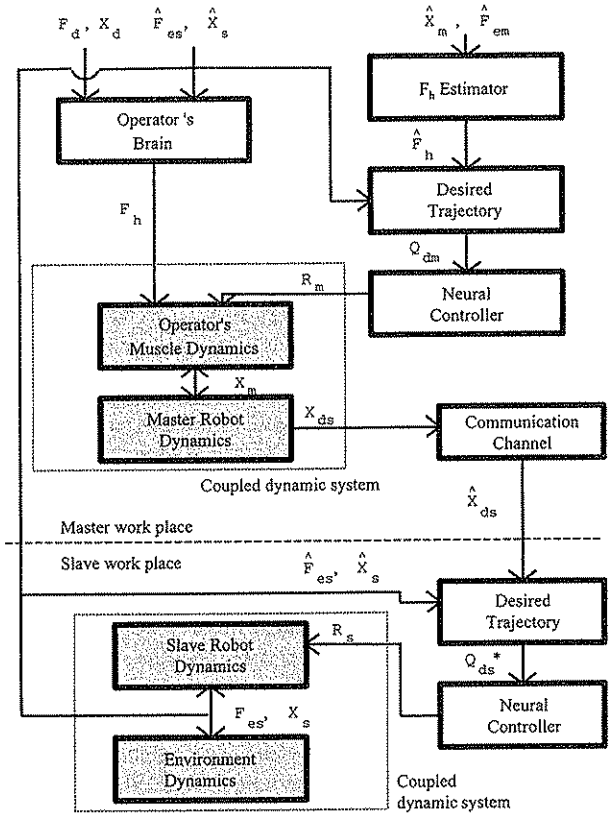

Fig. 10 Teleoperator with neural controllers, where ˆ denotes variables corrupted by noise

with the one described in Section 3.1. The controllers presented in the following Sections are capable to operate under all circumstances, and cope with much larger modeling errors at both robots. Moreover, our goal was to replace the initial computed torque controller with an improved one, since the desired trajectory following is not perfect, even with minimum modeling uncertainty. With errors up to 50% in the parameters of the last link the system becomes unstable. The reader is referred to Section 3.4.1 for simulation results on the response of the original system to computational delays and modeling errors.

3.3 Teleoperator control

Several neurocontrol architectures have been proposed in the literature, most of them stemming from previous classical control methods. Two comprehensive surveys can be found in [17], [44]. A comparison in the context of robotic control was reported in [26]. For the teleoperator system presented here, the variation of Model Reference Adaptive Control (MRAC) of Lee *et al.* [41] was selected (**Fig. 9**). The desired performance of each robot is determined in the form of Generalized Impedance. The resulting sets of differential equations can be considered as a model for the robot. Therefore the MRAC seems to be the most appropriate choice for use within the teleoperator.

As mentioned in Section 3.1 the desired performance for both robots in the original architecture of Lee and Lee is determined by the Generalized Impedance. The most straightforward way to incorporate the neural controller is to first calculate an ideal trajectory according to the GI and then use the NNs to force the robots follow it. The standard computed torque controller functions in a similar way. The difference is that there the GI is explicitly incorporated in the control torque generation formulas, whereas now a two-stage approach is followed.

Figure 10 shows in block diagram form the overall teleoperator architecture with the new controllers. This architecture realizes a two-level hierarchy. At the higher level, the operator and GI blocks determine the desired trajectories "thinking" in Cartesian coordinates. At the lower level, the controllers try to impose the ideal performance by calculating suitable control inputs. Starting from the slave side, the "Desired Trajectory Block" generates the command to the controller according to the master command and the GI parameters. Since the next block uses variables in the joint space, the inverse kinematics are also part of this block. The Neural Controller produces the control torque for the coupled dynamic system "slave robot + environment." In order to improve the system performance, the latter and the controller can be enclosed in a local control loop, with smaller sampling period than the overall system. However, simulations revealed that this is not necessary, as long as the communication channel allows high transmission rates. Furthermore, one can isolate the Desired Trajectory generator from the slave side position x_s and/or force f_{es}, by using for the GI calculations not the actual but the desired variables, which are local in the block. To use such a "desired" f_{es} a model of the environmental dynamics has to be known or estimated, or a filter for the measured force has to be used. In this way the "ideal" trajectory produced is not influenced by the slave tracking error, which can spoil the smoothness of the desired trajectory. The control loop is closed through the operator and the master. Although this approach relies on the good tracking capabilities of the controller, simulation results testing it were excellent.

On the master side the situation is more complicated. Two choices for the generation of the desired trajectory are possible: viz., equations (3) and (5). In the ideal case, the use of either would give the same results. Note that they both represent the GI, with the difference that the second is applied to the dual dynamic system: "operator's arm + master." In our opinion the second is the best choice, since

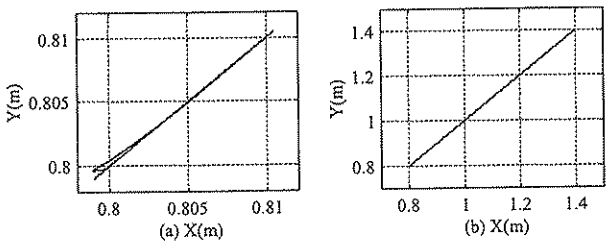

Fig. 11 Slave trajectory with MLPNNs: (a) Task A, representing the results without and with initial learning phase by the solid and dashed lines, where this simulation was performed with slightly different NN structure [1]—the structure used in the other simulations behaved almost perfectly to the same task (initial error < 1 [mm]), nevertheless this figure is presented to show what could happen with inadequate training or larger modeling errors; (b) Task C

the complete dynamics of the controlled part of the system are taken into account. If (3) is used, then the desired performance will be imposed exclusively on the robot, ignoring the internal dynamics of the human. In other words, the "ideal" desired trajectory generated would not be based on the human intentional force f_h, but on an indirect observation of it (f_{em}), i.e. on the result of its application to a dynamic system (the operator's arm). The other choice copes with all the passive parts of the couple "operator + master." It results from the fact that the human arm is also something to be controlled, considering f_h rather than f_{em} as the local system input. However, f_h is not easily measurable[†], whereas f_{em} is.

To overcome this problem an estimator of the human arm dynamics was incorporated in the control scheme. Lee and Lee [9] model the human arm in detail, and derive the simplified model mentioned in Section 2. In the latter, there is only one set of unknown parameters (matrix M_h), whose estimation is trivial. Even the detailed model is not difficult to identify. Since the parameters of the model change according to the person's fatigue and intention, an NN adopted online would be a suitable estimator. If it is preferred, then the block labeled "f_h estimator" in Fig. 10 should be renamed "f_{em} filter." In the following, this estimator is treated as a "black-box" which always gives the perfect f_h.

There are two other major difficulties concerning the master controller. First, it has to work in parallel with the operator, who—in contrast to the environment in the slave side—is not passive. Thus it acts only upon a part of the system, so that perfect performance cannot be guaranteed just by its design, unlike the slave side controller. Second, the master's function is to transmit to the operator the feel of the force and (together with the visual system) the feel of the movement at the slave side, which are the fundamental stimuli for the generation of f_h. Therefore the performance of the controller has to be excellent right from the start, so that the operator won't be misled. Ideally, its presence has to be unnoticed by the operator, who has to feel the GI trajectory.

Regarding these difficulties, the MLPNN achieves satisfactory performance after a short (1 [sec]) training phase performed automatically, during which their is no interaction with the operator. Early simulations indicated that right after the modeling error appears (e.g. after picking up an object) the robot can even move at first towards the wrong direction (**Fig. 11**a), even if we assume that the operator "understands" correctly f_{ref}.[†] Within a few milliseconds the neurocontroller performance—i.e. the accuracy of the model—is greatly improved and the robot starts moving along the desired path. This small error can be minimized by first ordering the robot to move along a sinusoidal course of small amplitude for a while and then stay still. In this way the NN learns coarsely the altered dynamics before the main movement begins. Although in later simulations with improved NN structure the system behaved almost perfectly to the same Task (initial error < 1 [mm]), this short additional phase is included for the integrity of the approach and for safety, in case larger modeling errors arise. With RBFNNs the adaptation to sudden parameters changes was found to be much faster, and so such a trick was not considered necessary.

With the incorporation of this phase, the training consists of three phases: a) An initial offline phase, during which the robot is trained on a nominal model and learns the basic nonlinear functions (this phase can be performed in a laboratory), b) A short, automatic, on-field but still offline phase whenever serious modeling errors (are expected to) arise, and c) A continuous online phase for perfectly tuning the parameters and following the desired trajectory. Using the terminology of classical identification, the first phase corresponds to the structure identification, the second to a coarse parameter estimation, and the last with the final parameters' tuning. This training phase, as a general concept, can be used in conjunction with any teleoperator task or control method. In summary, it should be noted that when designing controllers for the master, the engineer has to be more careful than for the slave side. With the exception of the points discussed above, the design concept of Lee and Lee was preserved without any other changes.

3.4 Simulation results

3.4.1 Original system response First some simulations were performed to check the response of the original algorithm of Lee and Lee. They were performed with two identical rigid-revolute-link manipulators of 2 DOFs acting as the master and slave arm. Their parameters, the modeling errors considered, as well as the overall teleoperator control system parameters are given in **Table 5**.

Table 6 shows the tasks for which the system was tested. The first three tasks are performed in free space. Tasks D and E are contact task. Both master and slave are initially stationary at free space, at the same position relative to their base frame. Then the operator tries to move the slave to position x_d and expects/estimates to feel a reaction force f_d (which is a scaled version of the actual $f_{es,d}$). This will actually be felt if he penetrates inside the object as much as the last column of Table 6 shows. The object in the contact tasks is modeled as an elastic, immobile "wall" of infinite

[†]It can be measured by implanted sensors or estimated from surface electrodes' measurements [10], [11].

[†]We still expect to assess by experiments how misleading such an "almost instant" discrepancy can be for a human operator. With proper training, the operator could learn to ignore transient errors.

Table 5 System parameters and modeling errors

Operator
$M_h = \begin{bmatrix} 0.15 & 0 \\ 0 & 0.15 \end{bmatrix}, B_{hu} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, K_{hu} = \begin{bmatrix} 7.0 & 0 \\ 0 & 7.0 \end{bmatrix}$
$B_{hf} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, K_{hf} = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}$
Master and Slave Dynamics
link 1: $m = 10, l = 1, l_c = 0.5, I = 0.8$
link 2: $m = 5, l = 1, l_c = 0.5, I = 0.45$
Units and symbols: SI units (m, kg, kg· m^2)
Symbols: m: mass, l: length, l_c: center of mass position from previous joint I: moment of inertia
Generalized Impedance
$M_{dm} = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}, B_{dm} = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, B_{fs} = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$
$M_{ds} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, B_{ds} = \begin{bmatrix} 8 & 0 \\ 0 & 8 \end{bmatrix}, K_{ds} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, K_{fs} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
Communication Channel and G_{rp}
$K_{sc} = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}, K_{cs} = \begin{bmatrix} 0.02 & 0 \\ 0 & 0.02 \end{bmatrix}, G_{rp} = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}$
Modeling Errors
Unless otherwise stated, 50% and 20% for the slave and master respectively on the mass and moment of inertia of the last link.

dimensions with $Z_e = \begin{bmatrix} 20 & 20 \\ 20 & 20 \end{bmatrix}$ [N/m]. A point on the border is $x_e = (0.8085\ [\text{m}], 0.8085\ [\text{m}])$, and a unity vector vertical to it is (0.7071, 0.7071). Its stiffness is equivalent to a spring vertical to its border with $k = 40$ [N/m].††

In the next figures, when referring to "ideal" response we mean the response computed directly by the GI equations, i.e. assuming they were perfectly imposed by the controller. For the "non-ideal" case the control input was first calculated and then applied to the robot dynamics after a time delay of one sampling period, in order to model the computational delay. All simulations were carried out with a sampling period of 1 [msec] on a PC with a 90 MHz Intel Pentium processor.

The ideal system response for Task B is shown in **Fig. 12**. However, in the same Figure it is shown that even if only a computation delay of a full sampling period (1 [msec]) is taken into account the slave will follow a sinusoidal course. Note that the results reported by Lee and Lee in [9], [22], [28] compare the system response with other teleoperator designs but not with the ideal (GI) performance. Furthermore, at Task A with 50% modeling errors only at the slave side (**Fig. 13**), the slave moves along a curved course and stops at a totally wrong position, probably because of the monitoring forces. With modeling errors at both robots, and on contact, the system becomes unstable. The above results justify the need for our new controller.

Table 6 Simulation tasks

Task	x_o	x_d	$f_{es,d}$	Penetration
A	(0.8, 0.8)	(0.8105, 0.8105)	–	–
B	(0.8, 0.8)	(0.85, 0.85)		
C	(0.8, 0.8)	(1.4, 1.4)	–	–
D	(0.8, 0.8)	(0.8105, 0.8105)	(5, 5)	0.1768
E	(0.8, 0.8)	(0.8105, 0.8105)	(1, 1)	0.0354

Variables with subscript "o" and "d" represent starting and goal values respectively. All distances are in meters, and forces in Newtons.

3.4.2 Response with MLP neural controllers Various sets of simulations were performed employing MLP neurocontrollers. Although the offline training data was forming a sparse grid over their whole input space, the NNs had no difficulty to adapt to such localized tasks. Figure 11 reveals the usefulness of the initial training phase. When the robot is ordered to move right after the modeling error appears (e.g. after picking up an object) it moves initially towards the wrong direction. Within a few milliseconds the neurocontroller performance—i.e. the accuracy of the model—is greatly improved and the robot starts moving along the desired path. This small error is minimized by first ordering the robot to move along a sinusoidal course of small ampli-

†† Yokokohji and Yoshikawa characterized a similar object as "relatively hard" [37].

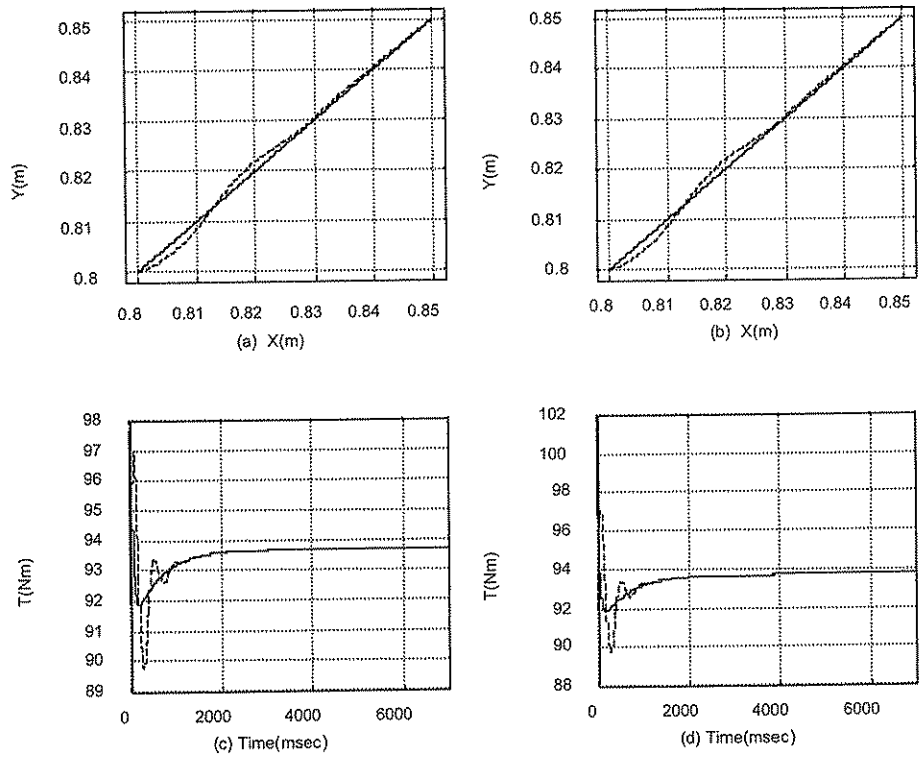

Fig. 12 Task B, representing the ideal system and the system with full period computation delay by the solid and dashed lines: (a) slave trajectory; (b) master trajectory; (c) input torque for the slave's first joint; and (d) input torque for the master's second joint

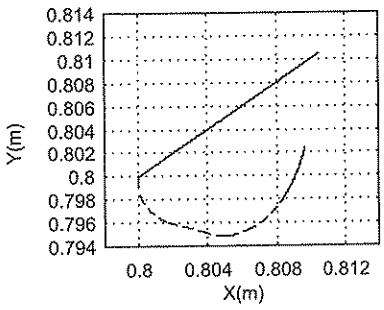

Fig. 13 Task A, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines

tude (1 mm at this simulation) for 0.5 [sec] and then stay still for another 0.5 [sec] (see Fig. 11b). This way the NN learns the altered dynamics before the main movement begins. This new training session will be used in the following figures. In the simulations we considered that modeling errors arise simultaneously at both sides. This is clearly a worst-case situation. It will be however shown that the proposed controllers can handle it.

Figure 14 shows the results for a small movement in free space. Note that the slave trajectory is a perfect line and that the time evolution of the distance from the target matches perfectly the ideal one, i.e. the one generated through GI. With the exception of the first few milliseconds, during which the subnets learn the new dynamics, the control torque has almost no chattering.† The peaks of the torque mark the beginning of the phases of training and movement.

Figure 15 (Task D) shows a typical trajectory when contact with the environment is commanded. In some simulations a small deviation from the linear course upon contact was observed. To minimize this deviation, the scaling factor K_{cs} was set to low values. Note also that due to the compromise imposed by the GI, the equilibrium position lies between the desired position x_d and the desired penetration (Table 6). It can be seen that after contact, the trajectory is slower and the equilibrium point is slightly different (< 1 [mm]) from the one expected by the GI. We have concluded that this is due to the contact shock, which obstructs the NN learning. Possible ways to minimize its effects are currently being investigated by the authors.

3. 4. 3 Response with RBF neural controllers In **Figs. 16** and **17** some representative results of the system with RBFNNs are shown. The trajectory is almost ideal, and no initial training phase is required as with MLPs. Much to our surprise, though, control chattering is more than MLPs, even more than sliding mode controllers discussed in the next paragraph.

A general remark, applying to both MLPNN and RBFNN, is that instability of the NN learning and consequently the whole teleoperator system can be caused if the online obtained training data is "too hard" for the NN, i.e. if the acceleration and speed are outside the limits used in the offline learning phase, or if the modeling error is excessively big. An area of "useable" training data has to be specified during the offline initialization phase, by both

†In [1] we reported some chattering, which later on disappeared by improving the structure of the MLPNN.

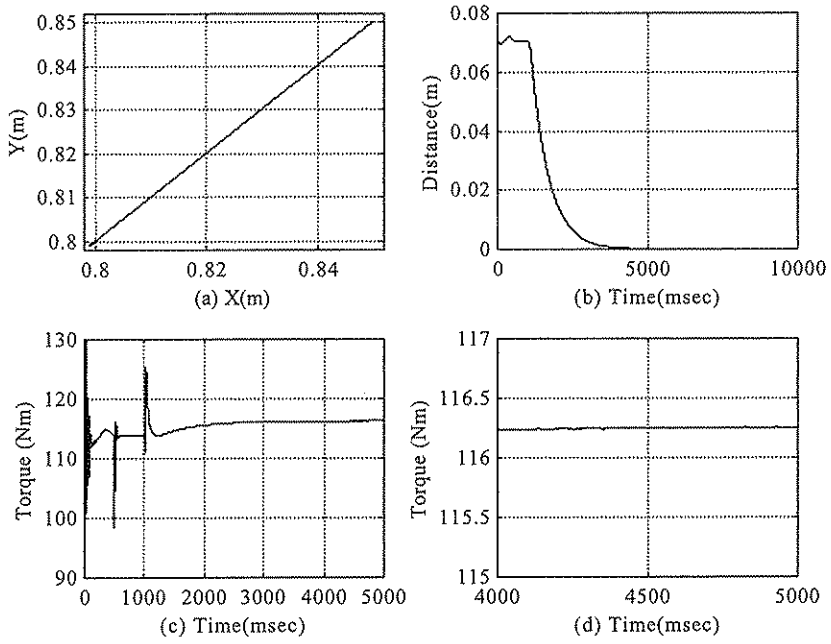

Fig. 14 Task B with MLPNNs, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

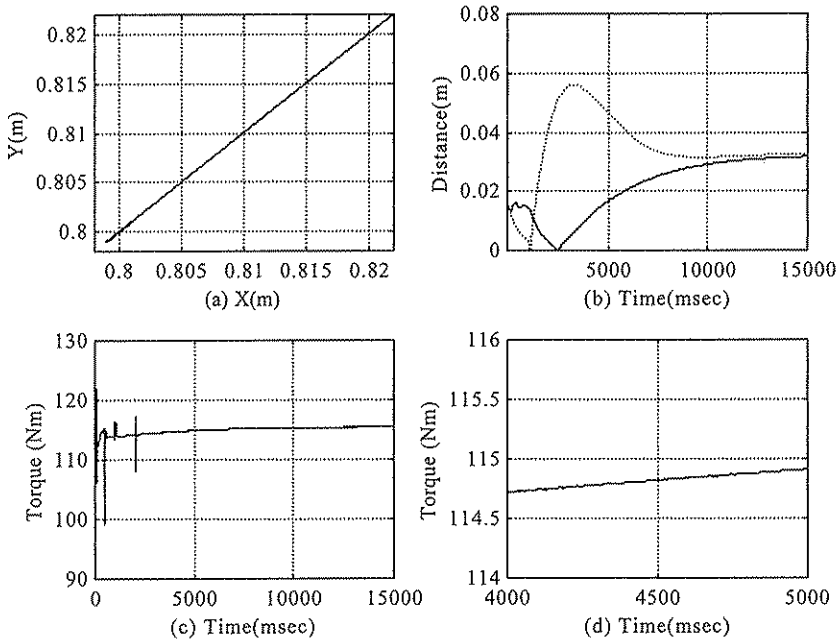

Fig. 15 Task D with MLPNNs, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

theoretical and trial-and-error methods. If "illegal" data is generated during the online movement, we have concluded that it is better to *not teach* this pair of data and use the current model for the generation of the control torque. A few milliseconds of "approximately correct" torque are preferable over instability! If the movement ordered continuously demands "illegal" NN input, then an alert has to be issued to the operator.

3.4.4 Response with sliding-mode controllers A sliding mode robust controller was also tested as an improvement of the control scheme of Lee and Lee [38]. Sliding controllers guarantee trajectory tracking under the presence of modeling uncertainties of known bounds and disturbances [6], [39], [45], [46]. The controller was incorporated

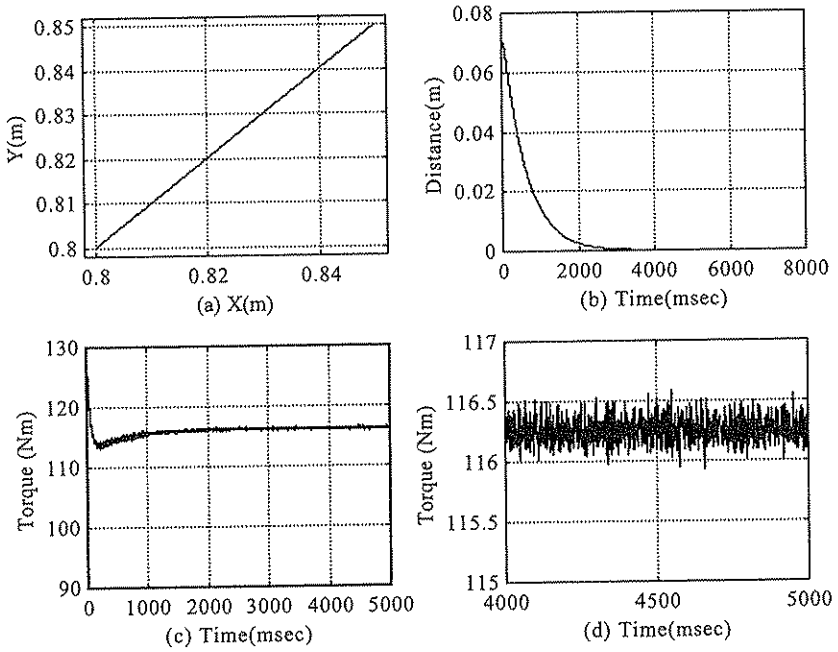

Fig. 16 Task B with RBFNNs, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

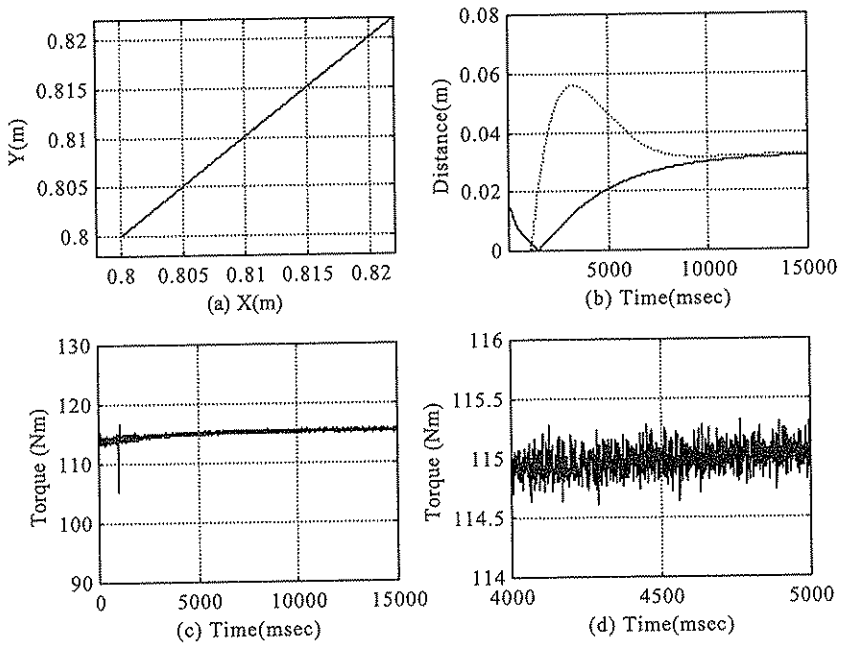

Fig. 17 Task D with RBFNNs, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

in the teleoperator system following the concept presented in Fig. 10a, where now the controller blocks represent the new type of controllers instead of NNs. A problem faced was how to handle force control, since the sliding mode method is primarily aimed at trajectory following. The most straightforward approach was to apply the two-step procedure also utilized for the neural controllers.

Figure 18 analyzes the system response for a movement in free space (Task B). Although the slave trajectory matches the ideal, some control chattering is visible. In contact tasks it becomes more intense. **Figure 19** shows the response for Task E. Note that although trajectory tracking is satisfactory during the movement, in the steady state there is a small discrepancy between the ideal and the ac-

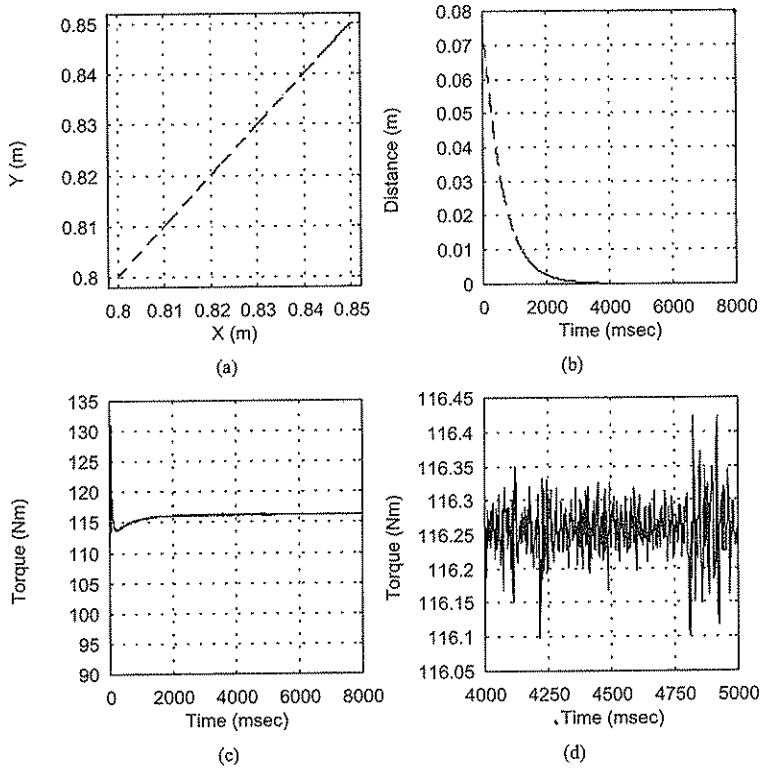

Fig. 18 Task B with sliding mode controllers, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

tual position, comprising to about 1 [mm] or 0.01 [N]. This error becomes insignificant for smaller desired forces. Increasing the coefficient K_{cs} led to a gradual elimination of it, but at the cost of amplification of control chattering.

3.4.5 Comparison of MLP, RBF and sliding-mode controllers Although a direct quantitative comparison between the methods is not easy, due to the differences in basic concepts and architectural parameters, a qualitative one is attempted in **Table 7**. Generally, all three methods produced practically ideal trajectories. Between the two NN families tested, RBF controllers performed better in some aspects: they never exhibited any initial trajectory overshoot, the "subnetwork confusion" was less and their offline training very easy. However, the control chattering was, surprisingly, worse than even the sliding controllers, and the MLPs had significantly less neurons. Therefore we would prefer the MLPNNs. A useful criterion is reported by Ziauddin and Zalzala [19] and relates to the hardware available: if serial hardware is used, then the MLPs are more suitable due to their small size. RBFs are fittest to be implemented on parallel hardware.

An engineer deciding which algorithm to implement, should also consider the classical control methods. As their representative in our work we used the sliding mode technique. As with the other members of this controller class, a sliding mode controller requires a mathematical nominal model of the robot dynamics as well as knowledge of the uncertainty bounds. On the other hand, NNs adapt quickly to a wide range of situations and parameters, and the output is calculated easier and faster. Finally, an inverse model of the system is identified. This can prove to be very useful to evaluate the situation and the objects encountered, much better than the "blind" compensation offered by sliding robust controllers. Therefore NNs are, in our opinion, more suitable for employment within the teleoperator system.

4. Conclusions and Future Work

This paper summarized and presented in a unified way some new results of the authors on robotic manipulator dynamics identification and control using NNs. The first part of the paper concentrated on the identification problem. The NN is divided in three subnetworks, each one corresponding to a part of the manipulator dynamics. A novel Error Distribution technique (termed HERD) was theoretically analyzed and supported by simulations. The second part of the paper employed the resulting NN to the teleoperator control problem. Two different teleoperator control architectures, a global and a local one, were proposed. The local one was then fully analyzed. Simulations with MLPNN and RBFNN, as well as with sliding-mode controllers, were reported. A final comparison slightly favored the MLPNNs.

In the future, as far as the teleoperator scheme is concerned, the investigation of other neural net architectures (e.g. recurrent architectures) are planned to be tested. NNs can also be exploited for other functions within a teleoperator system equally successfully. They can be used to model the environment, the operator's arm and his nervous system

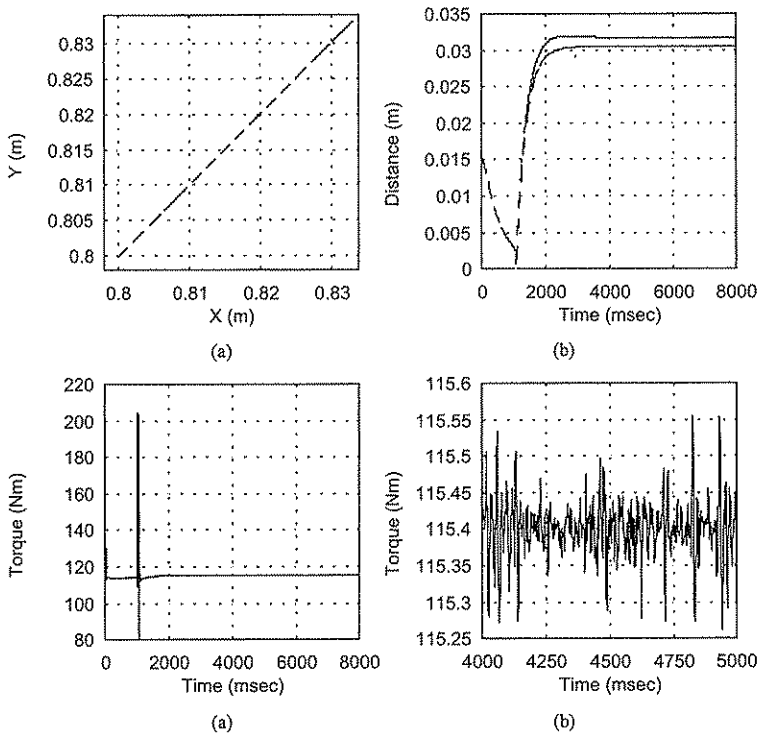

Fig. 19 Task E with sliding mode controllers, representing the ideal system and the system with modeling error at the slave by the solid and dashed lines: (a) slave trajectory; (b) distance from the target; (c) input torque at the slave's 1st link; and (d) input torque at the slave's 1st link

Table 7 Comparison of control methods

	MLP	RBF	Sliding mode
size of network/complexity	small	big	big
offline learning/mathematical computation	long (hours)	short (minutes)	long (hours)
online learning: confusion error	bigger	smaller	no learning
online learning: total error	very small	very small	no learning
guaranteed convergence	no	no	yes
robot trajectory	excellent	excellent	excellent
control chattering	insignificant	big ($o(1)$ Nm)	medium ($o(0.1)$ Nm)

response. In addition, some of the autonomous functions assigned to the slave can be accomplished by NNs, e.g. preserving the orientation or programming the moves of the end effector, local path planning, and obstacle avoidance. We also consider applying the partitioned neurocontroller as an underlying linearizing controller within the Neuro-predictive Teleoperation Scheme, introduced in [10], [11]. All these are fruitful areas for future research.

The effect of time delays in the communication channel was not analyzed in this paper. In order to simulate the stochastic nature of the communication channel and thus make the simulations more realistic, random delay will be introduced in future explorations. The effect of time delays in teleoperation and robustifying techniques to compensate for them are discussed in [10].

We also plan to further investigate the form of the HERD functions and generally $\boldsymbol{R}$ in (5) to suitably modify the search direction of the gradient descent algorithm, i.e. develop "manipulator-specific" or generally "plant-specific" training algorithms.

References

[1] S. G. Tzafestas, P. A. Prokopiou, and C. S. Tzafestas, "Robust telemanipulator control using a partitioned neural network architecture," in *Proc. of 1997 IEEE Int. Conf. on Neural Networks (ICNN '97)*, Houston, Texas, June, 1997.

[2] S. G. Tzafestas, P. A. Prokopiou, and C. S. Tzafestas, "Telemanipulator neurocontrol using multiple RBF networks," in *Proc. of 12th IEEE Int. Symp. on Intelligent Control (ISIC '97)*, Istanbul, Turkey, July 1997, pp. 257–262.

[3] S. G. Tzafestas and P. A. Prokopiou, "Error distribution to partitioned neural controllers identifying the robot dynamic model," in *Proc. of the 2nd Int. Symp. on Intelligent Automation and Control (ISIAC'98)*, Anchorage, Alaska, May 1998, paper No. 140.

[4] J. Sjoberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[5] S. G. Tzafestas, A. E. Krikochoritis, and C. S. Tzafestas, "Robust and adaptive control of biped-robot walking," in *Proc. of 1st Mobile Robotics Technology for Health Care Services Research Network (MobiNet) Symposium*, Athens, Greece, May 1997, pp. 271–286.

[6] S. G. Tzafestas, M. Raibert, and C. S. Tzafestas, "Robust sliding mode control applied to a 5-link biped robot," *J. of Intelligent and Robotic Systems*, vol. 15, no. 1, pp. 67–133, 1996.

[7] C. S. Tzafestas, P. A. Prokopiou, and S. G. Tzafestas, "Path planning and control of a cooperative three-robot system manipulating large objects," *J. of Intelligent and Robotic Systems*, vol. 22, no. 2, pp. 99–116, 1998.

[8] G. Hirzinger, "Towards a new robot generation for space, terrestrial and medical applications," in *Proc. of Second ECPD Int. Conf. on Advanced Robotics, Intelligent Automation and Active Systems*, Vienna, Austria, September 1996.

[9] S. Lee and H. S. Lee, "Modeling, design and evaluation of advanced teleoperator control systems with short time delay," *IEEE Trans. on Robotics and Automation*, vol. 9, no. 5, pp. 607–623, October 1993.

[10] P. Prokopiou, W. S. Harwin, and S. G. Tzafestas, "Exploiting a human arm model for fast intuitive and time-delays-robust telemanipulation," in *Advances in Manufacturing: Decision, Control and Information Technology*, S. G. Tzafestas, Ed. Berlin, Germany: Springer, 1999, pp. 255–266.

[11] P. A. Prokopiou, W. S. Harwin, and S. G. Tzafestas, "A novel scheme for human-friendly and time-delays robust neuropredictive teleoperation," *J. of Intelligent and Robotic Systems*, vol. 25, pp. 311–340, August 1999.

[12] A. Guez and J. Selinsky, "Neurocontroller design via supervised and unsupervised learning," *J. of Intelligent and Robotic Systems*, vol. 2, pp. 307–335, 1989.

[13] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki, "Hierarchical neural network model for voluntary movement with application to robotics," *IEEE Control Systems Magazine*, pp. 8–16. April 1988.

[14] F. L. Lewis, K. Liu, and A. Yesildirek, "Neural net robot controller with guaranteed tracking performance," *IEEE Trans. on Neural Networks*, vol. 6, no. 3, pp. 703–715, May 1995.

[15] F. L. Lewis, K. Liu, and A. Yesildirek, "Multilayer neural net robot tracking control," *IEEE Trans. on Neural Networks*, vol. 7, no. 2, pp. 388–399, March 1996.

[16] T. D. Sanger, "Neural networks learning control of robot manipulators using gradually increasing task difficulty," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 3, pp. 323–333, June 1994.

[17] S. G. Tzafestas, "Neural networks in robot control," in *Artificial Intelligence in Industrial Decision Making, Control and Automation*, S. G. Tzafestas and H. B. Verbruggen, Eds. Dordrecht, The Netherlands: Kluwer, 1995, pp. 327–328.

[18] S. T. Venkataraman, S. Gulati, J. Bahren, and N. Toomarian, "A neural network based identification of environments models for compliant control of space robots," *IEEE Trans. on Robotics and Automation*, vol. 9, no. 5, pp. 685–697, October 1993.

[19] S. M. Ziauddin and A. M. S. Zalzala, "Model-based compensation and comparison of neural network controllers for uncertainties of robotic arms," *IEE Proceedings on Control Theory and Applications*, vol. 142, no. 5, pp. 501–507, September 1995.

[20] Y. Yokokohji, A. Ogawa, H. Hasunuma, and T. Yoshikawa, "Operation modes for cooperating with autonomous functions in intelligent teleoperation systems," in *Proc. of 1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, Georgia, vol. III, 1993, pp. 510–515.

[21] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Trans. on Automatic Control*, vol. 34, no. 5, pp. 494–501, May 1989.

[22] S. Lee and H. S. Lee, "Design of optimal time delayed teleoperator control systems," in *Proc. of 1994 IEEE Int. Conf. on Robotics and Automation*, San Diego, California, U.S.A., 1994, pp. 3252–3258.

[23] G. Niemeyer and J.-J. E. Slotine, "Towards force reflecting teleoperation over the internet," in *Proc. of 1998 IEEE Int. Conf. on Robotics and Automation*, Leuven, Belgium, May 1998, pp. 1909–1915.

[24] P. Coiffet, "Robotics and virtual reality techniques: A new look on man-machine relations," in *Proc. of 1st Mobile Robotics Technology for Health Care Services Research Network (MOBINET) Symposium*, Athens, Greece, May 1997, pp. 11–25.

[25] D. H. Cha, H. S. Cho, and S. Kim, "Design of a force reflection controller for telerobot systems using neural network and fuzzy logic," *J. of Intelligent and Robotic Systems*, vol. 16, pp. 1–24, 1996.

[26] P. A. Prokopiou, C. S. Tzafestas, E. S. Tzafestas, and S. G. Tzafestas, "Neural network robust-adaptive telemanipulator control: comparison with sliding-mode control," *Syst. Anal. Modell. Simul.*, vol. 33, no. 3, pp. 259–294, 1998.

[27] S. Lee, "Intelligent sensing and control for advanced teleoperation," *IEEE Control Systems Magazine*, pp. 19–28, June 1993.

[28] S. Lee and H. S. Lee, "An advanced teleoperator control system: Design and evaluation," in *Proc. of 1992 IEEE Int. Conf. on Robotics and Automation*, Nice, France, 1992, pp. 859–864.

[29] T. B. Sheridan, "Telerobotics," *Automatica*, vol. 25, no. 4, pp. 487–507, 1989.

[30] A. A. Kobrinski and A. E. Kobrinski, *Bras Manipulateurs des Robots*. Moscow, Russia: Editions Mir, 1989 (in French).

[31] H. Kazerooni, T.-I. Tsai, and K. Hollerbach, "A controller design framework for telerobotic systems," *IEEE Trans. on Control Systems Technology*, vol. 1, no. 1, pp. 50–62, March 1993.

[32] J. E. Colgate, "Robust impedance shaping telemanipulation," *IEEE Trans. on Robotics and Automation*, vol. 9, no. 4, pp. 374–384, August 1993.

[33] B. Hannaford, "A design framework for teleoperators with kinesthetic feedback," *IEEE Trans. on Robotics and Automation*, vol. 5, no. 4, pp. 426–434, August 1989.

[34] D. A. Lawrence, "Stability and transparency in bilateral teleoperation," *IEEE Trans. on Robotics and Automation*, vol. 9, no. 5, pp. 624–637, October 1993.

[35] R. M. Satava, "The modern medical battlefield: Sequitur on advanced medical technology," *IEEE Robotics and Automation Magazine*, vol. 4, no. 3, pp. 21–25, September 1994.

[36] S. E. Salcudean, N. M. Wong, and R. L. Hollis, "Design and control of a force reflecting teleoperator system with magnetically levitated master and wrist," *IEEE Trans. on Robotics and Automation*, vol. 11, no. 6, pp. 844–857, December 1995.

[37] Y. Yokokohji and T. Yoshikawa, "Bilateral control of master-slave manipulators for ideal kinesthetic coupling—Formulation and experiment," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 5, pp. 605–619, October 1994.

[38] S. G. Tzafestas and P. A. Prokopiou, "Compensation of teleoperator uncertainties with a sliding mode controller," *J. of Robotics and Computer Integrated Manufacturing*, vol. 13, no. 1, pp. 9–20, January 1997.

[39] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Reading, MA: Addison-Wesley, 1989.

[40] S. G. Tzafestas (Ed.), *Intelligent Robotic Systems*. New York, NY: Marcel Dekker, 1991, Chapter 10.

[41] T. H. Lee, W. K. Tan, and M. H. Ang, "A neural network control system with parallel adaptive enhancements applicable to nonlinear servomechanisms," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 3, pp. 269–276, June 1994.

[42] S. G. Tzafestas, G. Stavrakakis, and A. Zagorianos, "Robot model reference adaptive control through lower/upper part dynamic decomposition," *J. of Intelligent and Robotic Systems*, vol. 1, pp. 163–184, 1988.

[43] S. Haykin, *Neural Networks*. New York, NY: Macmillan College Publishing, 1994.

[44] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural networks for control systems—A survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[45] C.-C. Kung and S.-C. Lin, "Fuzzy controller sesign: A sliding mode approach," in *Fuzzy Reasoning in Information Decision and Control Systems*, S. G. Tzafestas and A. N. Venetsanopoulos, Eds. Dordrecht, The Netherlands: Kluwer, 1994, pp. 278–306.

[46] J.-J. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[47] S. P. Chan, "Comments on: 'A neural network compensator for uncertainties of robotics manipulators', " *IEEE Trans. on Industrial Electronics*, vol. 42, no. 2, pp. 217–218, April 1995.

[48] T. C. Hsia and S. Jung, "A simple alternative to neural network control scheme for robot manipulators," *IEEE Trans. on Industrial Electronics*, vol. 42, no. 4, pp. 414–416, August 1995.

Biographies

Spyros G. Tzafestas is a full Professor, the Director of the Institute of Communications and Computer Systems (ICCS), the Signals, Control and Robotics Division, and the Intelligent Robotics and Automation Laboratory (IRAL) of the National Technical University of Athens (NTUA). He holds Ph.D. and D.Sc. in Control and Automation, and received Honorary Doctorates from the International University (D.Sc.(Hon.)) and of the Technical University of Munich (Dr.-Ing.E.h.). He is a fellow of IEEE (N.Y.) and IEE (London), and member of ASME (N.Y.), New York Academy of Sciences, IMACS (Rutgers, N.J.) and SIRES (Brussels). He is also a member of IFAC SECOM and MIM TCs, a project evaluator of national and international projects (USA, Canada, Italy, Hong-Kong, Japan), and a project coordinator of national and EU projects in the fields of robotics, CIM and IT (ESPRIT, BRITE-EURAM, TIDE, INTAS, SOCRATES, EUREKA, GROWTH etc.). He published 30 research books, 60 book chapters, over 700 journal and conference technical papers. He is an Editor-in-Chief of the Journal of Intelligent and Robotics Systems and of the book series "Microprocessor-Based and Intelligent Systems Engineering" (Kluwer). He has served as an organizer of several international conferences (IEEE, IFAC, IMACS, IASTED, SIRES etc.). His current research interests include control, robotics and CIM.

Platon A. Prokopiou received the Diploma in Electrical Engineering (1996) and the Ph.D. in Engineering (2000) from National Technical University of Athens, Greece. He is currently with the Intelligent Robotics and Automation Laboratory, Dept. of Electrical Engineering, National Technical University of Athens, Greece. His research interests include Robotics (especially Teleoperation, Human Arm and Operator Modeling for Teleoperation, Visual Servoing, Cooperative Robot Control, Mobile Robots, Artificial Life), Neural Networks, and Fuzzy Logic for Control.

Costas S. Tzafestas holds a Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece, and a DEA and Ph.D. degrees in Robotics from the Universite Pierre et Marie Curie (Paris 6), France. He is currently a research associate in the Institute of Informatics and Telecommunications at the National Center for Scientific Research "Demokritos," Athens. His research interests include virtual reality interfaces, haptic feedback, human/machine interaction, and telerobotics. He has also worked on robust, adaptive and neural control with applications in walking robots and cooperating manipulators. He is a member of the IEEE and of the Greek Technical Chamber.