Adaptive Neural Network Control of Robot Manipulators in Task Space

Shuzhi S. Ge, *Member, IEEE*, C. C. Hang, *Senior Member, IEEE*, and L. C. Woon

***Abstract*— In this paper, adaptive neural network control of robot manipulators in the task space is considered. The controller is developed based on a neural network modeling technique which neither requires the evaluation of inverse dynamical model nor the time-consuming training process. It is shown that, if Gaussian radial basis function networks are used, uniformly stable adaptation is assured, and asymptotically tracking is achieved. The controller thus obtained does not require the inverse of the Jacobian matrix. In addition, robust control can be easily incorporated to suppress the neural network modeling errors and the bounded disturbances. Numerical simulations are provided to show the effectiveness of the approach.**

***Index Terms*—Adaptive control, neural network, robot manipulator, task space.**

I. Introduction

ROBOT manipulators have become increasingly important in the field of flexible automation. Through the years, considerable research effort has been made in their controller design. In order to achieve accurate trajectory tracking and good control performance, a number of control schemes have been developed. Computed torque control is one of the most intuitive schemes, which relies on the exact cancellation of the nonlinear dynamics of the manipulator system. Such a scheme has the disadvantage of requiring the exact dynamic model. Furthermore, the payload of the robot manipulator may vary during its operation, which may not be necessarily known in advance. To overcome these problems, adaptive control strategies for robot manipulators have been developed and have attracted the interest of many researchers, as shown in [1] and [2], for example. These adaptive control methods have the advantage, in general, of requiring no *a priori* knowledge of unknown parameters, such as the mass of the payload. Learning control schemes [3] have also been developed, which improve the performance of the system when the same motion is performed repeatedly, so that learning can take place. A drawback of such a control technique is that generally it is only applicable to operations which are repetitive.

Recently, some developments have been made in the use of neural networks for the control of robot manipulators [4]–[7]. In general, neural network control design is done in two steps. Firstly, a neural network is used to approximate the dynamic model of the system. This approximation is usually carried out off-line and then, when a sufficiently accurate approximation is obtained, an appropriate control strategy using this approximation can be constructed. This approach has been shown to work well for many systems. However, it does not have any built-in capability to handle changes in the system. This is where incorporation of adaptive control is useful. Some recent works have successfully achieved this by using a suitable neural network to directly parameterize the control law [8], [9]. This leads to an overall closed-loop system with good stability properties. While most of the neural network controllers require the evaluation of inverse dynamic model, as well as the time-consuming training process, it is eliminated in the approaches proposed in [10] and [11]. The neural networks can simply be initialized to zero by assuming no knowledge about the system. Besides that, the controller is robust and easy for real-time implementation.

In this paper, the control method presented in [11] is further extended to the task space or the so-called Cartesian space. To apply robot manipulators to a wide class of tasks, it will be necessary to control not only the position of the end-effector, but also the force exerted by the end-effector on the object. By designing the control law in task space, force control can be easily formulated. Most controllers proposed thus far for adaptive manipulator tracking in the task space require some sort of inverse of the Jacobian matrix (see [2], for example). However, it is time-consuming and quite difficult to obtain the inverse of the Jacobian matrix. Moreover, it is prone to difficulties due to the kinematic singularities. By directly parameterizing the control law, we eliminate the need for the inverse of the Jacobian matrix.

This paper is organized as follows. In Section II, the Ge–Lee (GL) matrix and its product operator are introduced for the stability analysis of neural networks. The problem of neural network approximation is briefly described in Section III. In Section IV, the neural network modeling of robots is discussed. The controller design is presented in Section V and followed by a simulation example in Section VI. Finally, some concluding remarks are given in section Section VII.

II. GL Matrix and Operator

In this section, the definition of GL matrix, denoted by $\{\cdot\}$, and its product operator "$\bullet$" [11], [12] are briefly discussed. Readers are referred to [12] for a detailed discussion on the motivation of using GL matrix. To avoid any possible confusion, $[\cdot]$ is used to denote the ordinary vector and matrix.

Let I_0 be the set of integers and $\theta_{kj}, \xi_{kj} \in R^{n_{kj}}$, where $n_{kj} \in I_0, k = 1, 2, \cdots, n, j = 1, 2, \ldots, n$. The GL row vector

Manuscript received November 12, 1996; revised December 14, 1996.

The authors are with the Department of Electrical Engineering, National University of Singapore, Singapore 119260.

Publisher Item Identifier S 0278-0046(97)07759-9.

$\{\theta_k\}$ and its transpose $\{\theta_k\}^T$ are defined in the following way:

$$\{\theta_k\} = \{\theta_{k1} \quad \theta_{k2} \quad \cdots \quad \theta_{kn}\}$$
$$\{\theta_k\}^T = \{\theta_{k1}^T \quad \theta_{k2}^T \quad \cdots \quad \theta_{kn}^T\}. \tag{2.1}$$

The GL matrix $\{\Theta\}$ and its transpose $\{\Theta\}^T$ are defined accordingly as

$$\{\Theta\} = \begin{Bmatrix} \theta_{11} & \theta_{12} & \cdots & \theta_{1n} \\ \theta_{21} & \theta_{22} & \cdots & \theta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{n1} & \theta_{n2} & \cdots & \theta_{nn} \end{Bmatrix} = \begin{Bmatrix} \{\theta_1\} \\ \{\theta_2\} \\ \cdots \\ \{\theta_n\} \end{Bmatrix} \tag{2.2}$$

$$\{\Theta\}^T = \begin{Bmatrix} \theta_{11}^T & \theta_{12}^T & \cdots & \theta_{1n}^T \\ \theta_{21}^T & \theta_{22}^T & \cdots & \theta_{2n}^T \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{n1}^T & \theta_{n2}^T & \cdots & \theta_{nn}^T \end{Bmatrix}. \tag{2.3}$$

For a given GL matrix $\{\Xi\}$,

$$\{\Xi\} = \begin{Bmatrix} \xi_{11} & \xi_{12} & \cdots & \xi_{1n} \\ \xi_{21} & \xi_{22} & \cdots & \xi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{n1} & \xi_{n2} & \cdots & \xi_{nn} \end{Bmatrix} = \begin{Bmatrix} \{\xi_1\} \\ \{\xi_2\} \\ \cdots \\ \{\xi_n\} \end{Bmatrix} \tag{2.4}$$

the GL product of $\{\Theta\}^T$, and $\{\Xi\}$ is an $n \times n$ matrix defined as

$$[\{\Theta\}^T \bullet \{\Xi\}] = \begin{bmatrix} \theta_{11}^T \xi_{11} & \theta_{12}^T \xi_{12} & \cdots & \theta_{1n}^T \xi_{1n} \\ \theta_{21}^T \xi_{21} & \theta_{22}^T \xi_{22} & \cdots & \theta_{2n}^T \xi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{n1}^T \xi_{n1} & \theta_{n2}^T \xi_{n2} & \cdots & \theta_{nn}^T \xi_{nn} \end{bmatrix}. \tag{2.5}$$

The GL product of a square matrix and a GL row vector is defined as follows. Let $\Gamma_k = \Gamma_k^T = [\gamma_{k1} \ \gamma_{k2} \cdots \gamma_{kn}], \gamma_{kj} \in R^{m \times n_{kj}}, m = \Sigma_{j=1}^n \ n_{kj}$, then we have

$$\Gamma_k \bullet \{\xi_k\} = \{\Gamma_k\} \bullet \{\xi_k\}$$
$$:= [\gamma_{k1}\xi_{k1} \quad \gamma_{k2}\xi_{k2} \quad \cdots \quad \gamma_{kn}\xi_{kn}] \in R^{m \times n}. \tag{2.6}$$

Note that the GL product should be computed first in a mixed matrix product. For instance, in $\{A\} \bullet \{B\}C$, the matrix $[\{A\} \bullet \{B\}]$ should be computed first, and then follow by the multiplication of $[\{A\} \bullet \{B\}]$ with matrix C.

III. Neural Network Approximation

In the field of control engineering, neural network is often used to approximate a given nonlinear function $f(y)$ up to a small error tolerance. The function approximation problem can be stated formally as follows.

Definition 3.1: Given that $f(y)$: $R^n \to R^m$ is a continuous function defined on the set $y \in R^n$, and $\hat{f}(W, y)$: $R^{l \times m} \times R^n \to R^m$ is an approximating function that depends continuously on the parameter matrix W and y, the approximation problem is to determine the optimal parameter W^* such that, for some metric (or distance function) d,

$$d(\hat{f}(W^*, y), f(y)) \leq \epsilon \tag{3.1}$$

for an acceptable small ϵ [13].

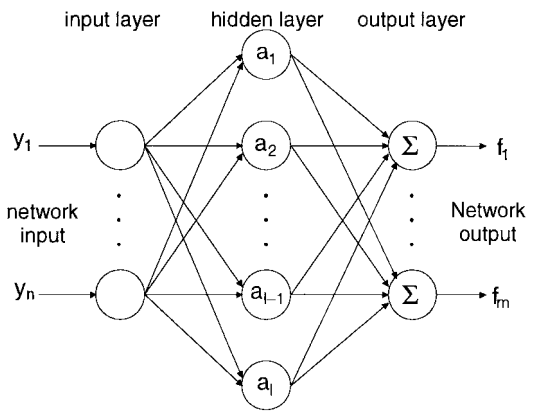

Fig. 1. Schematic diagram of RBF neural network.

In this paper, Gaussian radial basis function (RBF) neural network is considered. It is a particular network architecture which uses l numbers of Gaussian function of the form

$$a_i(y) = \exp\left(-\frac{(y-\mu_i)^T(y-\mu_i)}{\sigma^2}\right) \tag{3.2}$$

where $\mu_i \in R^n$ is the center vector and $\sigma^2 \in R$ is the variance. As shown in Fig. 1, each Gaussian RBF network consists of three layers: the input layer, the hidden layer that contains the Gaussian function, and the output layer. At the input layer, the input space is divided into grids with a basis function at each node defining a receptive field in R^n. The output of the network $\hat{f}(W, y)$ is given by

$$\hat{f}(W, y) = W^T a(y) \tag{3.3}$$

where $a(y) = [a_1(y) \ a_2(y) \quad \cdots \quad a_l(y)]^T$ is the vector of basis function. Note that only the connections from the hidden layer to the output are weighted.

Gaussian RBF network has been quite successful in representing the complex nonlinear function. It has been shown that a linear superposition of Gaussian radial basis function gives an optimal mean square approximation to an unknown function which is infinitely differentiable and the values of which are specified by a finite set of points in R^n [14]. Furthermore, it has been proven that any continuous functions, not necessary infinitely smooth, can be uniformly approximated by a linear combination of Gaussians [15], [16].

IV. Neural Network Modeling of Robot Manipulators

Consider a nonredundant rigid manipulator with n joints, the dynamics of which can be written as

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau \tag{4.1}$$

where $q \in R^n$ is the vector of joint variables, $\tau \in R^n$ is the vector of joint torques supplied by the actuators, $D(q) \in R^{n \times n}$ is the symmetric positive definite inertia matrix, $C(q, \dot{q}) \in R^{n \times n}$ is the Coriolis, and centrifugal matrix, $G(q) \in R^n$ is the vector of gravitational forces.

Usually, the manipulator task specification is given relative to the end-effector. Hence, it is natural to attempt to derive the control algorithm directly in the task space, rather than in the

joint space. Denote the end-effector position and orientation in the task space by $x \in R^n$. The task space dynamics can then be written as [17]

$$D_x(q)\ddot{x} + C_x(q,\dot{q})\dot{x} + G_x(q) = F_x \tag{4.2}$$

where

$$D_x(q) = J^{-T}(q)D(q)J^{-1}(q) \tag{4.3}$$

$$C_x(q,\dot{q}) = J^{-T}(q)(C(q,\dot{q}) - D(q)J^{-1}(q)\dot{J}(q))J^{-1}(q) \tag{4.4}$$

$$G_x(q) = J^{-T}(q)G(q) \tag{4.5}$$

$$F_x = J^{-T}(q)\tau \tag{4.6}$$

and $J(q) \in R^{n\times n}$ is the configuration-dependent Jacobian matrix, which is assumed to be nonsingular in the finite work space Ω. The above dynamic equation has the following useful structural properties, which can be exploited to facilitate the controller design in next section.

Property 1: The inertia matrix $D_x(q)$ is symmetric and positive definite.

Property 2: Matrix $N := \dot{D}_x(q) - 2C_x(q,\dot{q})$ is skew-symmetric if $C(q,\dot{q})$ is defined by the Christoffel symbols [18].

It is observed that both $D_x(q)$ and $G_x(q)$ are functions of q only; hence, static neural networks are sufficient to model them. Assume that $d_{xkj}(q)$ and $g_{xk}(q)$ can be modeled as

$$d_{xkj}(q) = \sum_l \theta_{kjl}\xi_{kjl}(q) + \varepsilon_{dkj}(q) = \theta_{kj}^T\xi_{kj}(q) + \varepsilon_{dkj}(q) \tag{4.7}$$

$$g_{xk}(q) = \sum_l \beta_{kl}\eta_{kl}(q) + \varepsilon_{gk}(q) = \beta_k^T\eta_k(q) + \varepsilon_{gk}(q) \tag{4.8}$$

where $\theta_{kjl}, \beta_{kl} \in R$ are the weights of the neural networks, $\xi_{kjl}(q), \eta_{kl}(q) \in R$ are the corresponding Gaussian basis functions with input vector q, and $\varepsilon_{dkj}(q), \varepsilon_{gk}(q) \in R$ are the modeling errors of $d_{xkj}(q)$ and $g_{xk}(q)$, respectively, and are assumed to be bounded. Whereas, for $C_x(q,\dot{q})$, a dynamic neural network of q and $\dot{q}$ is needed to model it. Assume that $c_{xkj}(q,\dot{q})$ can be modeled as

$$\begin{aligned} c_{xkj}(q,\dot{q}) &= \sum_l \alpha_{kjl}\zeta_{kjl}(z) + \varepsilon_{ckj}(z) \\ &= \alpha_{kj}^T\zeta_{kj}(z) + \varepsilon_{ckj}(z) \end{aligned} \tag{4.9}$$

where $z = [q^T\dot{q}^T]^T \in R^{2n}, \alpha_{kjl} \in R$ is the weights, $\zeta_{kjl}(z) \in R$ is the corresponding Gaussian basis function with input vector z, and $\varepsilon_{ckj}(z)$ is the modeling error of $c_{xkj}(q,\dot{q})$, which is also assumed to be bounded.

Therefore, the task space dynamics of the manipulators can be described as

$$D_x(q)\ddot{x} + C_x(q,\dot{q})\dot{x} + G_x(q) = F_x \tag{4.10}$$

with

$$d_{xkj}(q) = \theta_{kj}^T\xi_{kj}(q) + \varepsilon_{dkj}(q) \tag{4.11}$$

$$c_{xkj}(q,\dot{q}) = \alpha_{kj}^T\zeta_{kj}(z) + \varepsilon_{ckj}(z) \tag{4.12}$$

$$g_{xk}(q) = \beta_k^T\eta_k(q) + \varepsilon_{gk}(q). \tag{4.13}$$

Using the GL matrix and its product operator introduced in Section II, we can write $D_x(q)$ as

$$D_x(q) = [\{\Theta\}^T \bullet \{\Xi(q)\}] + E_D(q) \tag{4.14}$$

where $\{\Theta\}$ and $\{\Xi(q)\}$ are the GL matrices with their elements being θ_{kj} and $\xi_{kj}(q)$, respectively, as defined in (2.2) and (2.4), and $E_D(q) \in R^{n\times n}$ is a matrix with its elements being the modeling errors $\varepsilon_{dkj}(q)$. Similarly, for $C_x(q,\dot{q})$ and $G_x(q)$, we have

$$C_x(q,\dot{q}) = [\{A\}^T \bullet \{Z(z)\}] + E_C(z) \tag{4.15}$$

$$G_x(q) = [\{B\}^T \bullet \{H(q)\}] + E_G(q) \tag{4.16}$$

where $\{A\}, \{Z(z)\}, \{B\}$, and $\{H(q)\}$ are the GL matrices and vectors with their elements being $\alpha_{kj}, \zeta_{kj}(z), \beta_k$, and $\eta_k(q)$, respectively. $E_C(z) \in R^{n\times n}$ and $E_G(q) \in R^n$ are the matrix and vector with their elements being the modeling errors $\varepsilon_{ckj}(z)$ and $\varepsilon_{gk}(q)$, respectively.

V. Controller Design

Let $x_d(t)$ be the desired trajectory in the task space and $\dot{x}_d(t)$ and $\ddot{x}_d(t)$ be the desired velocity and acceleration. Define

$$e(t) = x_d(t) - x(t) \tag{5.1}$$

$$\dot{x}_r(t) = \dot{x}_d(t) + \Lambda e(t) \tag{5.2}$$

$$r(t) = \dot{x}_r(t) - \dot{x}(t) = \dot{e}(t) + \Lambda e(t) \tag{5.3}$$

where Λ is a positive definite matrix. With the following lemma, the stability of e and $\dot{e}$ can be concluded by studying r.

Lemma 5.1: Let $e(t) = h(t) * r(t)$, where "$*$" denotes convolution product and $h(t) = L^{-1}(H(s))$ with $H(s)$ is an $n \times n$ strictly proper, exponentially stable transfer function. Then, $r \in L_2^n$ implies that $e \in L_2^n \cap L_\infty^n, \dot{e} \in L_2^n, e$ is continuous, and $e \to 0$ as $t \to \infty$. If, in addition, $r \to 0$ as $t \to \infty$, then $\dot{e} \to 0$ [19].

Denote the estimate of $(\cdot)$ by $\hat{(\cdot)}$, and define $\tilde{(\cdot)} = (\cdot) - \hat{(\cdot)}$. Hence, $\{\hat{\Theta}\}, \{\hat{A}\}$, and $\{\hat{B}\}$ represent the estimates of the true parameter matrices $\{\Theta\}, \{A\}$ and $\{B\}$ of (4.14)–(4.16), respectively.

Consider a general controller of the form

$$\begin{aligned} F_x = &[\{\hat{\Theta}\}^T \bullet \{\Xi(q)\}]\ddot{x}_r + [\{\hat{A}\}^T \bullet \{Z(z)\}]\dot{x}_r \\ &+ [\{\hat{B}\}^T \bullet \{H(q)\}] + Kr + k_s\,\mathrm{sgn}(r) \end{aligned} \tag{5.4}$$

where $K \in R^{n\times n} > 0$ and $k_s > \|E\|$, with $E = E_D(q)\ddot{x}_r + E_C(z)\dot{x}_r + E_G(q)$. The first three terms of the control law are the model-based control, whereas the Kr term gives the proportional derivative (PD) type of control. Note that the PD control is effectively introduced to the control law through the definition of r given in (5.3). The last term in the control law is added to suppress the modeling errors of the neural networks. From (5.4), it is clear that the controller does not require the inverse of the Jacobian matrix. In real-time implementation, the control input τ that must be applied to the joint actuators can be computed using (4.6), as $\tau = J^T(q)F_x$.

Applying the control law from (5.4) to the manipulator dynamics in (4.10) and using (5.3) yields the tracking error equation

$$\begin{aligned} D_x(q)\dot{r} + C_x(q,\dot{q})r + Kr + k_s\,\mathrm{sgn}(r) \\ = [\{\tilde{\Theta}\}^T \bullet \{\Xi(q)\}]\ddot{x}_r + [\{\tilde{A}\}^T \bullet \{Z(z)\}]\dot{x}_r \\ + [\{\tilde{B}\}^T \bullet \{H(q)\}] + E. \end{aligned} \tag{5.5}$$

The stability property of the system (5.5) is given by the following theorem.

Theorem 5.1: For the closed-loop system (5.5), if $K > 0, k_s > \|E\|$ and the parameters are updated by

$$\dot{\hat{\theta}}_k = \Gamma_k \bullet \{\xi_k(q)\}\ddot{x}_r r_k \tag{5.6}$$

$$\dot{\hat{\alpha}}_k = Q_k \bullet \{\zeta_k(z)\}\dot{x}_r r_k \tag{5.7}$$

$$\dot{\hat{\beta}}_k = N_k \eta_k(q) r_k \tag{5.8}$$

where $\Gamma_k = \Gamma_k^T > 0, Q_k = Q_k^T > 0$ and $N_k = N_k^T > 0$, and $\hat{\theta}_k$ and $\hat{\alpha}_k$ are the column vectors with their elements being $\hat{\theta}_{kj}$ and $\hat{\alpha}_{kj}$, respectively, then, $\hat{\theta}_k, \hat{\alpha}_k, \hat{\beta}_k \in L_\infty$, and $e \in L_2^n \cap L_\infty^n, e$ is continuous, e and $\dot{e} \to 0$ as $t \to \infty$.

Proof: Consider the nonnegative scalar function V as

$$\begin{aligned} V = \tfrac{1}{2}\, r^T D_x(q) r + \tfrac{1}{2} \sum_{k=1}^{n} \tilde{\theta}_k^T \Gamma_k^{-1} \tilde{\theta}_k \\ + \tfrac{1}{2} \sum_{k=1}^{n} \tilde{\alpha}_k^T Q_k^{-1} \tilde{\alpha}_k + \tfrac{1}{2} \sum_{k=1}^{n} \tilde{\beta}_k^T N_k^{-1} \tilde{\beta}_k \end{aligned} \tag{5.9}$$

where Γ_k, Q_k, and N_k are dimensional compatible symmetric positive-definite matrices. Computing the derivative of (5.9) along (5.5) and simplifying yields

$$\begin{aligned} \dot{V} = r^T (D_x(q)\dot{r} + C_x(q,\dot{q})r) + \sum_{k=1}^{n} \tilde{\theta}_k^T \Gamma_k^{-1} \dot{\tilde{\theta}}_k \\ + \sum_{k=1}^{n} \tilde{\alpha}_k^T Q_k^{-1} \dot{\tilde{\alpha}}_k + \sum_{k=1}^{n} \tilde{\beta}_k^T N_k^{-1} \dot{\tilde{\beta}}_k \end{aligned} \tag{5.10}$$

where the property of skew-symmetric has been used. Substituting the error equation (5.5) and noting that

$$r^T[\{\tilde{\Theta}\}^T \bullet \{\Xi(q)\}]\ddot{x}_r = \sum_{k=1}^{n} \{\tilde{\theta}_k\}^T \bullet \{\xi_k(q)\}\ddot{x}_r r_k$$

$$r^T[\{\tilde{A}\}^T \bullet \{Z(z)\}]\dot{x}_r = \sum_{k=1}^{n} \{\tilde{\alpha}_k\}^T \bullet \{\zeta_k(z)\}\dot{x}_r r_k$$

$$r^T[\{\tilde{B}\}^T \bullet \{H(q)\}] = \sum_{k=1}^{n} \tilde{\beta}_k^T \eta_k(q) r_k$$

equation (5.10) becomes

$$\begin{aligned} \dot{V} = -r^T K r - k_s r^T \mathrm{sgn}(r) + \sum_{k=1}^{n} \{\tilde{\theta}_k\}^T \bullet \{\xi_k(q)\}\ddot{x}_r r_k \\ + \sum_{k=1}^{n} \{\tilde{\alpha}_k\}^T \bullet \{\zeta_k(z)\}\dot{x}_r r_k \\ + \sum_{k=1}^{n} \tilde{\beta}_k^T \eta_k(q) r_k + r^T E + \sum_{k=1}^{n} \tilde{\theta}_k^T \Gamma_k^{-1} \dot{\tilde{\theta}}_k \\ + \sum_{k=1}^{n} \tilde{\alpha}_k^T Q_k^{-1} \dot{\tilde{\alpha}}_k + \sum_{k=1}^{n} \tilde{\beta}_k^T N_k^{-1} \dot{\tilde{\beta}}_k. \end{aligned} \tag{5.11}$$

Substituting the parameter update laws given in (5.6)–(5.8) into (5.11), with $k_s > \|E\|$, yields

$$\dot{V} \le -r^T K r \le 0. \tag{5.12}$$

a) From (5.12) and, since $K > 0$, it follows that $r \in L_2^n$. Consequently, from Lemma 5.1, $e \in L_2^n \cap L_\infty^n, e$ is continuous and $e \to 0$ as $t \to \infty$, and $\dot{e} \in L_2^n$.

b) Since $\dot{V} \le -r^T K r \le 0$, it follows that $0 \le V(t) \le V(0), \forall t \ge 0$. Hence, as $V(t) \in L_\infty$, this implies that $r, \tilde{\theta}_k, \tilde{\alpha}_k$ and $\tilde{\beta}_k \in L_\infty$, i.e., $\hat{\theta}_k, \hat{\alpha}_k$ and $\hat{\beta}_k \in L_\infty$.

By observing that $r \in L_2^n, x_d, \dot{x}_d, \ddot{x}_d \in L_\infty^n$, and $\{\Xi(q)\}, \{Z(z)\}$, and $\{H(q)\}$ are bounded basis functions, we can conclude that $\dot{r} \in L_\infty^n$ from (5.5). Therefore, r is uniformly continuous. The proof is completed using the following fact [19]: r is uniformly continuous, and $r \in L_2^n \Rightarrow r \to 0$ as $t \to \infty$.

Hence, $\dot{e} \to 0$ as $t \to \infty$. Q.E.D.

Remarks:

1) If Γ_k and Q_k are defined as

$$\Gamma_k = \begin{bmatrix} \Gamma_{k1} & 0 & \cdots & 0 \\ 0 & \Gamma_{k2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Gamma_{kn} \end{bmatrix}$$

$$Q_k = \begin{bmatrix} Q_{k1} & 0 & \cdots & 0 \\ 0 & Q_{k2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q_{kn} \end{bmatrix} \tag{5.13}$$

where $\Gamma_{kj}, Q_{kj}, 1 \le j \le n$, are multidimensional compatible matrix blocks, then the parameter adaptation (5.6) and (5.7) can be expressed as

$$\dot{\hat{\theta}}_{kj} = \Gamma_{kj}\xi_{kj}(q)\ddot{x}_{rj} r_k \tag{5.14}$$

$$\dot{\hat{\alpha}}_{kj} = Q_{kj}\zeta_{kj}(z)\dot{x}_{rj} r_k. \tag{5.15}$$

2) The center μ and the variance σ^2 of the RBF's are fixed arbitrarily. These parameters, however, can be fixed by adding an off-line learning step to find different clusters of their centers and variances such that $D_x(q), C_x(q,\dot{q})$, and $G_x(q)$ are better represented.

3) The presence of the $\mathrm{sgn}(\cdot)$ function inevitably introduces chattering, which is undesirable. To alleviate this problem, a boundary layer can be used as suggested in [2].

VI. Simulation Example

For illustrative purposes, a planar two-link manipulator used extensively in the literature is considered in our simulation study. The dynamic equation of the robot can be written as

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{6.1}$$

where

$$D(q) = \begin{bmatrix} m_1 + m_2 + 2m_3\cos q_2 & m_2 + m_3\cos q_2 \\ m_2 + m_3\cos q_2 & m_2 \end{bmatrix}$$

$$C(q,\dot{q}) = \begin{bmatrix} -m_3\dot{q}_2\sin q_2 & -m_3(\dot{q}_1 + \dot{q}_2)\sin q_2 \\ m_3\dot{q}_1\sin q_2 & 0.0 \end{bmatrix}$$

$$G(q) = \begin{bmatrix} m_4 g\cos q_1 + m_5 g\cos(q_1 + q_2) \\ m_5 g\cos(q_1 + q_2) \end{bmatrix}$$

and m_i are the parameters of interest given by $M = P + p_l L$ with

$$M = [m_1 \quad m_2 \quad m_3 \quad m_4 \quad m_5]^T$$
$$P = [p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5]^T$$
$$L = [l_1^2 \quad l_2^2 \quad l_1 l_2 \quad l_1 \quad l_2]^T$$

and p_l is the payload, l_1 and l_2 are the lengths of link 1 and link 2, respectively, and P is the parameter vector of the robot itself. The Jacobian matrix $J(q)$ is known as

$$J(q) = \begin{bmatrix} -l_1 \sin(q_1) - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix}. \tag{6.2}$$

The true parameters of the robot used for simulation are

$$P = [1.66 \quad 0.42 \quad 0.63 \quad 3.75 \quad 1.25]^T \text{ kg} \cdot \text{m}^2$$

and each link has the length of 1 m. Assume that we have no knowledge about the system and there is no payload, i.e.,

$$\hat{P} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]^T \text{ kg} \cdot \text{m}^2, \qquad p_l = 0.0 \text{ kg}.$$

The desired trajectory in the Cartesian space is chosen as

$$x_{d1}(t) = 1.0 + 0.2\cos(\pi t); \quad x_{d2}(t) = 1.0 + 0.2\sin(\pi t)$$

which represents a circle of radius 0.2 m, and its center is located at $(x_1, x_2) = (1.0, 1.0)$ m. The robot is initially rested with its end-effector positioned at the center of the circle, i.e.,

$$x(0) = [1.0 \quad 1.0] \text{ m}; \qquad \dot{x}(0) = [0.0 \quad 0.0] \text{ m/s}.$$

For each element of $D_x(q)$ and $G_x(q)$, a 100-node static neural network is used, whereas, for each element of $C_x(q, \dot{q})$, a 200-node dynamic neural network is chosen. The values of μ and σ^2 are fixed at 0.0 and 10.0, respectively. The gains for the controller are chosen as

$$K = \text{diag}[10.0]; \qquad \Lambda = \text{diag}[5.0].$$

The sliding mode control term is excluded by setting $k_s = 0$, to show the effectiveness of the neural network controller. In order to test load disturbance rejection of the controller, a payload $p_l = 0.5$ kg was put on at time $t = 4.0$ s.

A. Case 1: Nonadaptive Control

Let us first study the control performance when the weights adaptations law of the neural network controller given by (5.6)–(5.8) is not activated. In this case, the resulting control action is effectively a simple PD control. Figs. 2–4 show the position and velocity tracking performances of the robot and the corresponding control signal. It can be observed from these results that the nonadaptive neural network control has a significant tracking error and cannot handle changes in the system, such as model approximation errors and load disturbances.

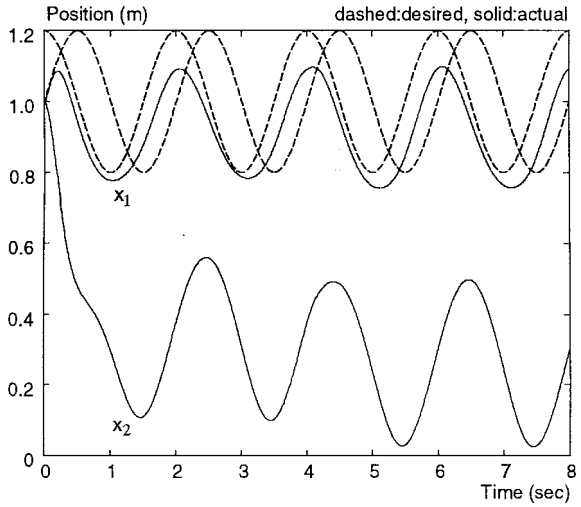

Fig. 2. Position tracking without adaptation.

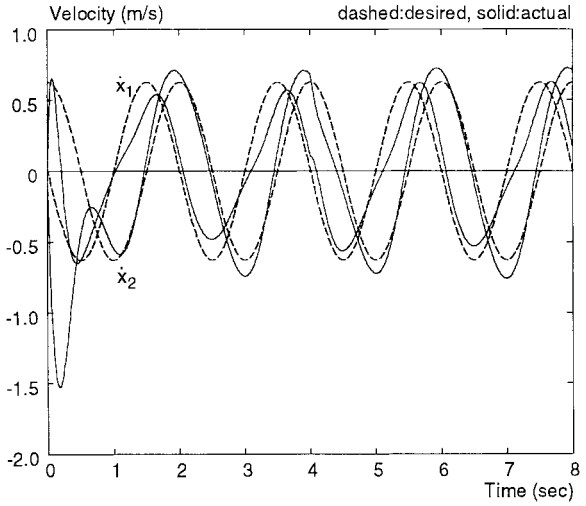

Fig. 3. Velocity tracking without adaptation.

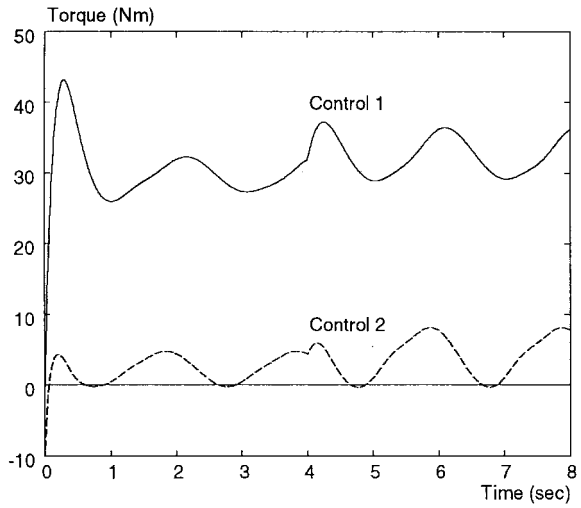

Fig. 4. Control signals without adaptation.

B. Case 2: Adaptive Control

In this case, the adaptation algorithms as given in (5.6)–(5.8) were activated with $\Gamma_k = \text{diag}[0.02], Q_k = \text{diag}[0.1]$ and

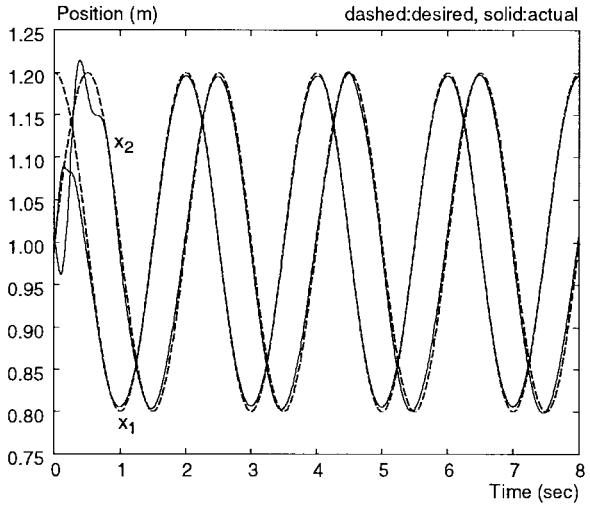

Fig. 5. Position tracking with adaptation.

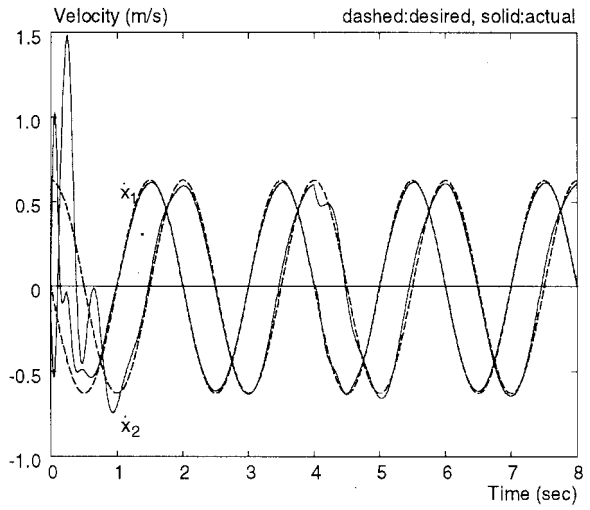

Fig. 6. Velocity tracking with adaptation.

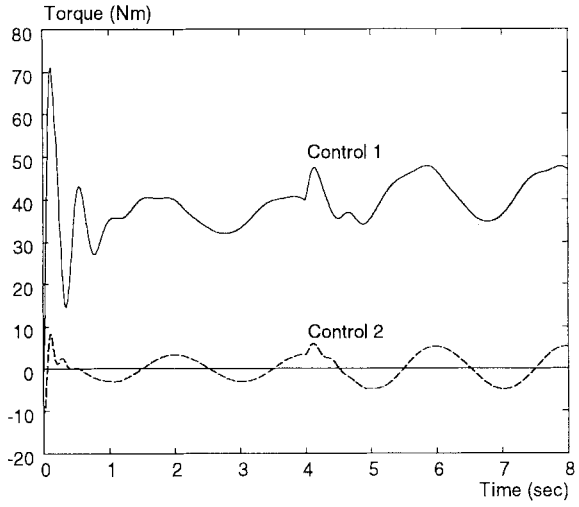

Fig. 7. Control signals with adaptation.

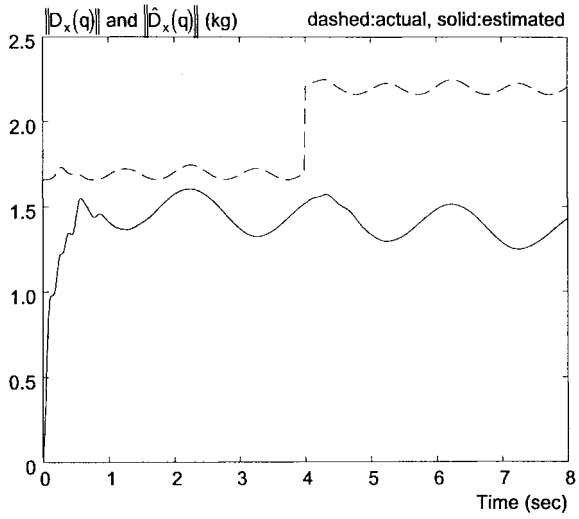

Fig. 8. Comparison of $||D_x(q)||$ with $||\hat{D}_x(q)||$ of adaptive neural network control.

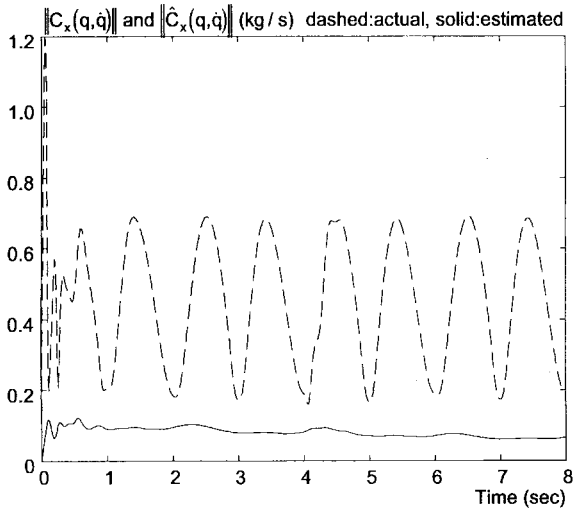

Fig. 9. Comparison of $||C_x(q,\dot{q})||$ with $||\hat{C}_x(q,\dot{q})||$ of adaptive neural network control.

$N_k = \text{diag}[5.0]$. The position and velocity tracking performance of the robot is plotted in Figs. 5 and 6, respectively, and the control input is given in Fig. 7. It can be seen that the tracking error is much smaller than the nonadaptive case because of the "learning" mechanism. The simulation results thus demonstrate that the proposed adaptive neural network control can effectively control the unknown nonlinear dynamic system and load disturbances. By adjusting the adaptation gain and other factors, such as the size of the networks, different tracking performance can be achieved. The variation of the elements of $D_x(q)$ and $\hat{D}_x(q), C_x(q,\dot{q})$ and $\hat{C}_x(q,\dot{q}), G_x(q)$ and $\hat{G}_x(q)$ are shown in Figs. 8–10. It is noted that both $\hat{D}_x(q)$ and $\hat{C}_x(q,\dot{q})$ do not converge to $D_x(q)$ and $C_x(q,\dot{q})$, even though $\hat{G}_x(q)$ does converge to $G_x(q)$. This is due to the fact that the desired trajectory is not persistently exciting, and this occurs quite often in real-world application.

VII. Conclusion

Adaptive control of rigid robot manipulators in the task space has been studied in this paper. The controller is developed based on the neural network modeling technique proposed in [11]. Unlike many neural network controllers

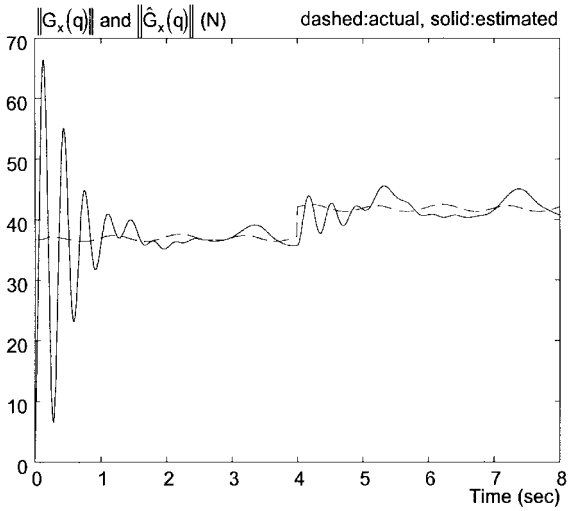

Fig. 10. Comparison of $||G_x(q)||$ with $||\hat{G}_x(q)||$ of adaptive neural network control.

proposed in the literature, inverse dynamical model evaluation is not required, and no time-consuming training process is necessary. It has been shown that, if Gaussian radial basis function networks are used, uniformly stable adaptation is assured, and asymptotically tracking is achieved. The controller has the advantage that the inverse of the Jacobian matrix is not required. In addition, the controller can be easily modified to achieve robustness to network modeling errors and bounded disturbance. Numerical simulations were also provided to demonstrate the performance of the controller.

REFERENCES

[1] J. J. Craig, P. Hsu, and S. S. Sastry, "Adaptive control of mechanical manipulators," *Int. J. Robotics Res.*, vol. 6, no. 2, pp. 16–28, 1987.
[2] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *Int. J. Robotics Res.*, vol. 6, no. 3, pp. 49–59, 1987.
[3] S. Arimoto, "Learning control theory for robot motion," *Int. J. Adapt. Control Signal Process.*, vol. 4, no. 6, pp. 543–564, 1990.
[4] W. T. Miller, F. H. Glanz, and I. G. Kraft, "Application of a general learning algorithm to control of a robotic manipulator," *Int. J. Robotics Res.*, vol. 6, no. 2, pp. 84–98, 1987.
[5] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback error learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, no. 3, pp. 251–265, 1988.
[6] T. Ozaki, T. Suzuki, T. Furuhashi, S. Okuma, and Y. Uchikawa, "Trajectory Control of robotic manipulators using neural networks," *IEEE Trans. Ind. Electron.*, vol. 38, pp. 195–202, June 1991.
[7] M. Saad, L. A. Dessaint, P. Bigras, and K. Al-haddad, "Adaptive versus neural network adaptive control: application to robotics," *Int. J. Adapt. Control Signal Process.*, vol. 8, no. 3, pp. 223–236, 1994.
[8] R. M. Sanner and J.-J. E. Slotine, "Gaussian networks for direct adaptive control," *IEEE Trans. Neural Networks*, vol. 3, pp. 837–863, Nov. 1992.
[9] E. Tzirkel-Hancock and F. Fallside, "A direct control method for a class of nonlinear systems using neural network," Cambridge Univ., Cambridge, U.K., Rep. CUED/F-INFENG/TR65, 1991.
[10] F. L. Lewis, K. Liu, and A. Yesildirek, "Neural net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Networks*, vol. 6, pp. 703–715, May 1995.
[11] S. S. Ge and C. C. Hang, "Direct adaptive neural network control of robots," *Int. J. Syst. Sci.*, vol. 27, no. 6, pp. 533–542, 1996.
[12] S. S. Ge, "Robust adaptive NN feedback linearization control of nonlinear systems," *Int. J. Syst. Sci.*, vol. 27, no. 12, pp. 1327–1338, 1996.
[13] J. R. Rice, *The Approximation of Functions*. Reading, MA: Addison-Wesley, 1964.
[14] T. Poggio and F. Girosi, "A theory of neural networks for approximation and learning," Artificial Intelligence Lab., MIT, Cambridge, MA, Memo. 1140, July 1989.
[15] F. Girosi and T. Poggio, "Networks and the best approximation property," Artificial Intelligence Lab., MIT, Cambridge, MA, Memo. 1164, Oct. 1989.
[16] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, pp. 1481–1497, Sept. 1990.
[17] F. L. Lewis, C. T. Abdallah, and D. M. Dawson, *Control of Robot Manipulators*. New York: Maxwell Macmillan, 1993.
[18] R. Ortega and M. W. Spong, "Adaptive motion control of rigid robots: A tutorial," *Automatica*, vol. 25, no. 6, pp. 877–888, 1989.
[19] C. Desoer and M. Vidyasagar, *Feedback Systems: Input-Output Properties*. New York: Academic, 1975.

Shuzhi S. Ge (S'90–M'92) received the B.Sc. degree in control engineering in 1986 from Beijing University of Aeronautics and Astronautics, Beijing, China, and the Ph.D. and DIC degrees in mechanical/electrical engineering in 1993 from Imperial College of Science, Technology and Medicine, University of London, London, U.K.

From May 1992 to June 1993, he was a Post-Doctoral Research Associate in the Department of Engineering, University of Leicester, Leicester, U.K. Since July 1993, he has been with the Department of Electrical Engineering, National University of Singapore, as a Lecturer. His current research interests are robust adaptive control, adaptive control of robots, neural network and fuzzy logic control, GA optimization, and real-time control systems.

C. C. Hang (S'70–M'73–SM'90) received the First Class Honors degree in electrical engineering from the University of Singapore in 1970 and the Ph.D. degree in control engineering from the University of Warwick, Warwick, U.K., in 1973.

From 1974 to 1977, he was a Computer and Systems Technologist with Shell Eastern Petroleum Company, Singapore, and Shell International Petroleum Company, The Netherlands. Since 1977, he has been with the National University of Singapore, serving in various positions, including Vice Dean of the Faculty of Engineering, Head of the Department of Electrical Engineering and, since October 1994, Deputy Vice Chancellor of the University. His major area of research is adaptive control, on which topic he has published one book and 150 international journal and conference papers. He is also the holder of four patents in this area. He was a Visiting Scientist at Yale University in 1983 and at Lund Institute of Technology in 1987 and 1992. Since March 1992, he has been the Principal Editor (Adaptive Control) of *Automatica*.

L. C. Woon received the B.Eng. degree with first class honors in electrical engineering in 1996 from the National University of Singapore, where he is currently working toward the M.Eng. degree.

His main research interests are in the areas of control of robot manipulators, neural network control, and real-time control systems.