On Adaptive Trajectory Tracking of a Robot Manipulator Using Inversion of Its Neural Emulator

Laxmidhar Behera, Madan Gopal, and Santanu Chaudhury

Abstract—**This paper is concerned with the design of a neuro-adaptive trajectory tracking controller. The paper presents a new control scheme based on inversion of a feedforward neural model of a robot arm. The proposed control scheme requires two modules. The first module consists of an appropriate feedforward neural model of forward dynamics of the robot arm that continuously accounts for the changes in the robot dynamics. The second module implements an efficient network inversion algorithm that computes the control action by inverting the neural model. In this paper, a new extended Kalman filter (EKF) based network inversion scheme is proposed. The scheme is evaluated through comparison with two other schemes of network inversion: gradient search in input space and Lyapunov function approach. Using these three inversion schemes the proposed controller was implemented for trajectory tracking control of a two-link manipulator. Simulation results in all cases confirm the efficacy of control input prediction using network inversion. Comparison of the inversion algorithms in terms of tracking accuracy showed the superior performance of the EKF based inversion scheme over others.**

I. Introduction

ROBOT manipulators are characterized by complex nonlinear dynamical structures with inherent unmodeled dynamics and unstructured uncertainties. These features make the designing of controllers for manipulators a difficult task in the framework of classical adaptive and nonadaptive control [1]–[4]. Artificial neural networks offer promising possibilities for providing better solutions to robot tracking problems, primarily because of their excellent capability to learn any complex mapping from training examples [5], [6]. Simulation and experimental results of a number of investigators such as Narendra and Parthasarathy [7], Miller *et al.* [8], Goldberg and Pearlmutter [9], Miyamoto *et al.* [10], Bassi and Bekey [11], Gomi and Kawato [12], and others have confirmed the potential of these networks in the area of dynamic modeling and control of nonlinear systems.

The existing literature on neural control of robot manipulators reveals that most of these applications are based on learning inverse dynamics of a robot arm. Some of these learning schemes of inverse dynamics can be summarized as follows. In direct inverse modeling of robot manipulators [8], [9], the network is trained using input–output data of the plant in a reverse fashion (i.e., joint angle $\theta \rightarrow$ input torque τ) directly. In the case of forward and inverse modeling [13], the forward model is learned by monitoring the input torque vector $\tau(t)$ and output joint position vector $\theta(t)$ of the manipulator. Next, the desired trajectory $\theta_{\mathrm{d}}(t)$ is fed to the inverse model to calculate the feedforward motor command $\tau_{ff}(t)$. The resulting error in the trajectory $\theta_{\mathrm{d}}(t) - \theta(t)$ is backpropagated through the forward model to calculate the command error, which is then used as the error signal for training the inverse model. The feedback error learning scheme of Miyamoto *et al.* [10] uses feedback torque as the error signal to train the inverse dynamic model of the robot arm. These inverse models compute the feedforward torque given a desired trajectory, and feedback stabilizing signal is actuated by a simple potential difference (PD) controller.

In our present work, a different approach is proposed where instead of training a separate network to learn the inverse dynamics, we directly perform the iterative inversion of a forward model of a robot arm to generate control action on-line. This approach avoids a separate scheme of providing a feedback stabilizing signal [8], [9], [14]. The motivating factor for this work is to see the feasibility of implementing direct inversion of a neural network to actuate on-line control signal, an approach that has got little attention in control literature. This approach also provides an alternative to backpropagation of utility as proposed by Werbos [15] and Nguyen and Widrow [16], which are basically off-line schemes.

Iterative inversion of multilayered neural networks based on gradient search in input space was first proposed by Linden and Kindermann [17]. This work shows that iterative inversion of a neural network model is possible by ascribing backpropagated errors in network output to errors in the network input signal. This technique has been extended to adaptive control of simple linear systems by Hoskins *et al.* [18]. Lee [19] has proposed a network inversion scheme based on the Lyapunov function approach in application to pattern recognition problems. Extension of this technique to on-line inversion of neural networks to predict control input is not reported in the literature. Keeping these facts in mind, in this paper we have explored the applications of the above two inversion schemes to robot tracking problems. In an attempt to search for a fast and robust inversion scheme, we have proposed an extended Kalman filter (EKF) based inversion algorithm for radial basis function network model of multi-input/multioutput (MIMO) systems. The proposed inversion scheme was motivated by the algorithm proposed by Iiguni *et al.* [20] and Scalero and Tepedelenlioglu [21] for training feedforward networks. Since EKF was found to be fast and efficient for training multilayered networks (MLN's), EKF

Manuscript received March 28, 1994; revised March 1, 1995 and May 10, 1996.

The authors are with the Department of Electrical Engineering, Indian Institute of Technology Delhi, Hauz Khas, New Delhi 110 016 India.

Publisher Item Identifier S 1045-9227(96)07459-0.

based inversion is expected to give better performance for on-line inversion applications.

The proposed controller based on each of the three different approaches of network inversion is implemented for a two-link robot manipulator. The forward dynamics of the manipulator is modeled by a radial basis function network (RBFN) network. The RBFN model is trained by input–output data generated in the robot workspace using a simple PD controller. Experiments were carried out to evaluate the efficiency of each inversion algorithm in terms of root mean square (rms) tracking errors by varying the initial conditions and changing the upper-bound on the number of iterations allowed per sampling interval to predict the control input. These experiments indicate that the performance of the EKF based inversion scheme is better than the other two. The present paper is organized in the following manner. Section II describes the nonlinear dynamic modeling using an RBFN and the corresponding learning mechanism. Three different schemes of network inversion are presented in Section III. The structure of the proposed robot trajectory controller is explained in Section IV. In Section V, we provide the simulation results by implementing a proposed neuro-adaptive controller for tracking control of the two-link manipulator. Finally, concluding remarks are given in Section VI.

II. RBFN Model of Forward Dynamics of Robot Arm

Consider a class of nonlinear discrete time dynamical systems described by

$$\boldsymbol{x}(k+1) = \boldsymbol{f}[\boldsymbol{x}(k), \boldsymbol{u}(k)] \tag{1}$$

where $\boldsymbol{x}(k) \in \Re^n$ and $\boldsymbol{u}(k) \in \Re^p$ represent, respectively, state and input vectors of the system at the kth sampling instant. The states of the system are assumed to be accessible and nonlinear function $\boldsymbol{f}(\cdot)$ is assumed to be unknown.

The unknown mapping $\boldsymbol{f}(\cdot)$ can be learned using either an MLN or RBF network. In MLN, activation of output layer neurons has a highly nonlinear relationship with the network parameters. Training algorithms based on recursive least squares methods [22] or the EKF approach [20], [21] are consequently computationally intensive. The popular back-propagation algorithm is prone to local minima trap and relatively slow in convergence. On the other hand, RBF network response is a linear function of its weights. This allows us to choose any of the least squares methods to train the network. In general, the learning in RBFN is fast. As we require the neural emulator to account for continuous changes in plant dynamics, RBFN is preferred over MLN.

A. Neural Modeling of $f(\cdot)$ Using RBFN

Fig. 1 shows an m-input/n-output RBF network. The ith output of such a network can be expressed as

$$\hat{x}_i = g_i(\boldsymbol{v}) = \sum_{j=1}^{\ell} \theta_{ij}\phi_j(\|\boldsymbol{v} - \boldsymbol{c}_j\|) \tag{2}$$

where $\boldsymbol{\nu} \in \Re^m$ is the network input vector; $\|\cdot\|$ denotes the Euclidean norm; $\boldsymbol{c}_j \in \Re^m, 1 \le j \le \ell$ are RBF centers; $\phi_j(\cdot)$ is

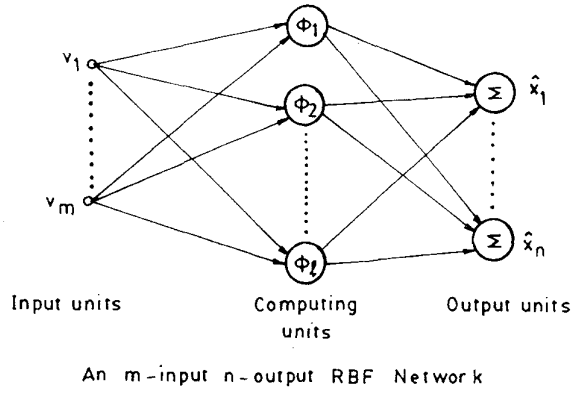


Fig. 1. An m-input n-output RBF network.

the jth activation function of hidden layer; $\theta_{ij}, 1 \le j \le \ell, 1 \le i \le n$ are the connection weights from hidden layer to output layer; and ℓ is the number of hidden units in the first layer. Typical choices for activation functions are Gaussian, thin-plate splines, etc. With any one of these activation functions RBF networks are capable of constructing reasonably good approximations of unknown functions [23], [24]. In fact, Powell [6] has shown that thin-plate-spline functions have good modeling capability. For our present application we have opted for the following thin-plate spline function:

$$\phi_j(d) = d^2 \log d. \tag{3}$$

We have made this choice because with this activation function, unlike a Gaussian function, we do not need to estimate the free parameter, i.e., width of the basis function [25]. RBF network parameters θ and $\boldsymbol{c}$ are required to be determined so that network response $\boldsymbol{g}(\cdot)$ can approximate the underlying dynamics $\boldsymbol{f}(\cdot)$ in (1). Define the input vector $\boldsymbol{v}(k)$ as

$$\boldsymbol{v}(k) = [\boldsymbol{x}(k)^T, \boldsymbol{u}(k)^T]^T. \tag{4}$$

Then the estimated states will be described as

$$\hat{\boldsymbol{x}}(k+1) = \boldsymbol{g}(\boldsymbol{v}(k), \boldsymbol{c}, \theta). \tag{5}$$

The estimation error $\boldsymbol{e}(k+1)$ is defined by

$$\begin{aligned}\boldsymbol{e}(k+1) &= \boldsymbol{x}(k+1) - \hat{\boldsymbol{x}}(k+1) \\ &= \boldsymbol{f}[\boldsymbol{x}(k), \boldsymbol{u}(k)] - \boldsymbol{g}[\boldsymbol{v}(k), \boldsymbol{c}, \theta].\end{aligned} \tag{6}$$

The RBFN has captured the dynamics for an optimal set of parameters θ and $\boldsymbol{c}$ if

$$\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) - \boldsymbol{g}(\boldsymbol{v}, \boldsymbol{c}, \theta)\| = \|\boldsymbol{e} < \epsilon \quad \text{for all } (\boldsymbol{x}, \boldsymbol{u}) \in \boldsymbol{D} \tag{7}$$

where ϵ is a suitably chosen constant and $\boldsymbol{D}$ is the region of operation in the input space.

B. RBFN Learning

A number of learning algorithms [19], [23], [26], [27] is available to train RBF networks. Here we will briefly summarize the learning mechanisms available for real-time applications. RBFN is trained in a two-stage process: 1) choosing radial centers and 2) adjusting weights.

The radial centers are chosen in such a manner that these centers suitably sample the network input domain and should be able to track the changing pattern of data. One of the practices is to use an n-means clustering technique to update the RBF centers as suggested by Moody and Darken [28]. In this work we have chosen these centers to be of uniform random distribution over the input space.

Because the response of the network is linear with respect to its weights, the recursive least squares methods may be used for adjusting weights. In the following we provide the recursive least squares algorithm (RLS) that has been employed in the present work.

The ith output of the RBFN described earlier can be written as

$$\hat{x}_i = \phi^T \theta_i, \qquad i = 1, \cdots, n \tag{8}$$

where $\phi \in \Re^\ell$ is the output vector of the hidden layer; $\theta_i \in \Re^\ell$ is the connection weight vector from the hidden units to the ith output unit. The weight update equations as per RLS algorithm are described as

$$\begin{aligned}\hat{\theta}_i(k) =& \hat{\theta}_k(k-1) + \boldsymbol{P}(k)\phi(k-1) \\ & \cdot [x_i(k) - \phi(k-1)^T \hat{\theta}_i(k-1)] \qquad (9)\\ \boldsymbol{P}(k) =& \boldsymbol{P}(k-1) - \boldsymbol{P}(k-1)\phi(k-1)[1 + \phi(k-1)^T \\ & \cdot \boldsymbol{P}(k-1)\phi(k-1)]^{-1}\phi(k-1)^T \boldsymbol{P}(k-1) \qquad (10)\end{aligned}$$

where $\boldsymbol{P}(k) \in \Re^{\ell \times \ell}$. Initial value $\boldsymbol{P}(0)$ is taken as $20\boldsymbol{I}$ for the present application and that of weight vector θ_i can be assigned to zero or small random numbers.

III. Network Inversion Algorithms

In this section, we present three different network inversion schemes which are useful for applications in control. The RBFN model [as given by (5)] represents a nonlinear mapping from m-dimensional input space to n-dimensional output space where $n < m (m = p + n)$. The objective of inverse operation on this model is to predict only p-inputs out of m number of total inputs. The remaining n inputs are known *a priori* (present system states). Thus the inverse mapping can be mathematically expressed as

$$\hat{\boldsymbol{u}}(k) = \boldsymbol{g}^{-1}[\boldsymbol{x}(k), \boldsymbol{x}^d(k+1), \boldsymbol{c}, \theta] \tag{11}$$

where $\boldsymbol{x}^d(k+1)$ is the desired output activation. This shows that basically we are performing inversion mapping $\boldsymbol{g}^{-1}$: $\Re^n \rightarrow \Re^p$ where $p < n$ for the N-link robot manipulator tracking control ($p = N, n = 2N$).

These inversion algorithms carry out the mapping given in (11) by updating input activation $\hat{\boldsymbol{u}}(k)$ iteratively until desired output activation is achieved or the number of iterations reaches a maximum, $t_{\max}$ within a sampling interval. This upper-bound $t_{\max}$, therefore, should be determined on the basis of the sampling interval and the computation time required per iteration. It is clear from this discussion that speed of convergence of the inversion algorithm can be measured by the number of iterations required to produce the desired output. The initial guess of the input activation $\hat{\boldsymbol{u}}(k)$ during each sampling interval is taken as the input activation $\hat{\boldsymbol{u}}(k-1)$ predicted in the previous sampling instant. For the case of first sampling interval, the initial guess is selected arbitrarily from the input space.

A. Gradient Search (GS) in Input Space

The iterative inversion of the RBF network can be carried out using the gradient descent algorithm as proposed by Linden and Kindermann [17]. The iterative rule is given as

$$\hat{u}_i^{t+1}(k) = \hat{u}_i^t(k) - \eta \frac{\partial E}{\partial u_i^t(k)} + \alpha[\hat{u}_i^t(k) - \hat{u}_i^{t-1}(k)] \tag{12}$$

where t refers to iterative step, η is the learning gain, and α is the momentum rate. The error function is given by $E = 1/2 \ \Sigma_{i=1}^n (x_i^d - \hat{x}_i)^2$. To prevent the input activation $\hat{\boldsymbol{u}}(k)$ from growing without limit, the input is considered as an output of a "pseudoneuron" with a limited output range [17].

For the RBF network shown in Fig. 1, with thin-plate-spline activation function, $\partial E/\partial u_i$ can be expressed as

$$\frac{\partial E}{\partial u_i} = -\sum_{j=1}^{n} (x_j^d - \hat{x}_j) \frac{\partial \hat{x}_j}{\partial u_i} \tag{13}$$

and

$$\begin{aligned}\frac{\partial \hat{x}_j}{\partial u_i} &= \sum_{k=1}^{\ell} \frac{\partial \hat{x}_j}{\partial \phi_k} \frac{\partial \phi_k}{\partial u_i} \\ &= \sum_{k=1}^{\ell} \theta_{jk}[1 + 2\log d(k)](u_i - c_{k,i}) \qquad (14)\end{aligned}$$

where $d(k) = \|\boldsymbol{v} - \boldsymbol{c}_k\|$.

B. Inverse Mapping Following Lyapunov Function (LF) Approach

The inverse mapping based on the Lyapunov function as a general means of achieving a recall process for selective attention as applicable to a pattern recognition process has been presented by Lee [19]. We adapt the same concept for control applications in the following way.

The Lyapunov function candidate V is chosen to be a quadratic error function in desired trajectories given by

$$V = \tfrac{1}{2} \tilde{\boldsymbol{x}}^T \tilde{\boldsymbol{x}} \quad \text{where} \quad \tilde{\boldsymbol{x}} = \boldsymbol{x}^d - \hat{\boldsymbol{x}}. \tag{15}$$

Here, $\boldsymbol{x}_d$ is the desired output activation and $\hat{\boldsymbol{x}}$ is the actual output activation of the RBFN model. The time derivative of the Lyapunov function is given by

$$\begin{aligned}\dot{V} &= \tilde{\boldsymbol{x}}^T \dot{\tilde{\boldsymbol{x}}} \\ &= -\tilde{\boldsymbol{x}}^T \frac{\partial \hat{\boldsymbol{x}}}{\partial \boldsymbol{u}} \dot{\boldsymbol{u}} \\ &= -\tilde{\boldsymbol{x}}^T \boldsymbol{J} \dot{\boldsymbol{u}} \qquad (16)\end{aligned}$$

where $\boldsymbol{J} = \partial \hat{\boldsymbol{x}}/\partial \boldsymbol{u}$; $n \times p$ Jacobian matrix.

Theorem 1: If an arbitrary initial input activation $\boldsymbol{u}(0)$ is updated by

$$\boldsymbol{u}*(t') = \boldsymbol{u}(0) + \int_0^{t'} \dot{\boldsymbol{u}}\, dt \tag{17}$$

where

$$\dot{\boldsymbol{u}} = \frac{\|\tilde{\boldsymbol{x}}\|^2}{\|\boldsymbol{J}^T\tilde{\boldsymbol{x}}\|^2}\boldsymbol{J}^T\tilde{\boldsymbol{x}} \tag{18}$$

then $\tilde{\boldsymbol{x}}$ converges to zero under the condition that $\dot{\boldsymbol{u}}$ exists along the convergence trajectory.

Proof: Substitution for $\dot{\boldsymbol{u}}$ from (18) into (16) we have

$$\dot{V} = -\|\tilde{\boldsymbol{x}}\|^2 \le 0 \tag{19}$$

where $\dot{V} < 0$ for all $\tilde{\boldsymbol{x}} \neq 0$ and $\dot{V} = 0$ iff $\tilde{\boldsymbol{x}} = 0$. Q.E.D.

The iterative input activation update rule based on (17) can be given as

$$\hat{\boldsymbol{u}}(t) = \hat{\boldsymbol{u}}(t-1) + \mu\dot{\hat{\boldsymbol{u}}}(t-1) \tag{20}$$

where μ is a small constant representing the update rate.

C. EKF-Based Inversion Algorithm

The EKF-based algorithm was proposed for training of MLN's by Iiguni *et al.* [20], Scalero and Tepedelenlioglu [21], and others. Here we propose a decoupled form of the inversion algorithm based on the EKF approach for RBF networks.

EKF is a method of estimating the state vector. Here, the unknown input vector $\boldsymbol{u}(k)$ is considered as the state vector to be estimated. The input vector of RBFN has m-components out of which n-inputs, i.e., the present state vector of the system $\boldsymbol{x}(k)$ are known at the kth sampling instant. Remaining p components $(\boldsymbol{u}(k))$ of the input vector $\boldsymbol{v}$ are estimated through the inversion process. During the iterative inversion of the network, RBFN parameters $\boldsymbol{c}$ and θ are held constant. So, RBFN output can be expressed as

$$\begin{aligned}\hat{\boldsymbol{x}}(k+1) &= \boldsymbol{g}[\boldsymbol{x}(k), \boldsymbol{u}(k), \boldsymbol{c}, \theta] \\ &= \boldsymbol{h}(\boldsymbol{u}(k)).\end{aligned} \tag{21}$$

The individual update of an input u_i can be decoupled from other updates if we assume that other $(p-1)$ inputs are known *a priori*. In other words, estimates of these $(p-1)$ inputs obtained in the previous iteration can be used for the update of the ith input in the present iteration. The inversion scheme presented here is based on this assumption.

Let the output vector and desired output vector of RBFN corresponding to the tth iteration and kth sampling instant be $\hat{\boldsymbol{x}}(k,t)$ and $\boldsymbol{x}^d(k+1)$, respectively. The RBFN can be expressed by the following nonlinear system equations as a function of the ith input (k has been dropped from the argument list for convenience):

$$u_i(t+1) = u_i(t) \tag{22}$$

$$\begin{aligned}\boldsymbol{x}^d &= \boldsymbol{h}[u_i(t)] + \xi(t) \\ &= \hat{\boldsymbol{x}}(t) + \xi(t).\end{aligned} \tag{23}$$

Here, $\xi(t)$ is assumed as a white noise vector with $n \times n$ covariance matrix $\boldsymbol{R}(t)$. The application of EKF to (22) and (23) gives the following real-time learning algorithm [29]:

$$\hat{u}_i(t) = \hat{u}_i(t-1) + \boldsymbol{K}_i(t)[\boldsymbol{x}^d - \hat{\boldsymbol{x}}(t)] \tag{24}$$

$$\boldsymbol{K}_i(t) = P_i(t-1)\boldsymbol{H}_i(t)^T[\boldsymbol{H}_i(t)P_i(t-1)\boldsymbol{H}_i(t)^T + \boldsymbol{R}_i(t)]^{-1} \tag{25}$$

$$P_i(t) = P_i(t-1) - P_i(t-1)\boldsymbol{K}_i(t)\boldsymbol{H}_i(t) \tag{26}$$

where $\boldsymbol{K}_i(t)$, the $(1 \times n)$ matrix, is called the Kalman gain and $\boldsymbol{H}_i(t)$, the $(n \times 1)$ matrix, is defined as

$$\boldsymbol{H}_i(t) = \left[\frac{\partial \boldsymbol{h}}{\partial u_i}\right]_{\boldsymbol{u}=\hat{\boldsymbol{u}}(t-1)}. \tag{27}$$

The jth element of $\boldsymbol{H}_i(t)$ can be expressed for the case of thin-plate-spline activation function as follows:

$$\begin{aligned}\boldsymbol{H}_{ij} &= \frac{\partial h_j}{\partial u_i} \\ &= \frac{\partial \hat{x}_j}{\partial u_i}.\end{aligned} \tag{28}$$

The expression for $\partial\hat{x}_j/\partial u_i$ is given in (14). During the inversion process as each output of the feedforward network is considered to be an independent function of control inputs only (23), the output covariance matrix $\boldsymbol{R}(t)$ can safely be assumed to be diagonal matrix $\lambda\boldsymbol{I}$. This assumption avoids the matrix inversion involved in (25). Applying the matrix inversion lemma, we have

$$\begin{aligned}&[\boldsymbol{H}_i(t)P_i(t-1)\boldsymbol{H}(t)^T + \boldsymbol{R}_i(t)]^{-1} \\ &\quad = \frac{1}{\lambda}\left[\boldsymbol{I} - \frac{P_i(t-1)\boldsymbol{H}_i(t)\boldsymbol{H}_i(t)^T}{\lambda + P_i(t-1)\boldsymbol{H}_i(t)^T\boldsymbol{H}_i(t)}\right].\end{aligned} \tag{29}$$

The inversion algorithm is simplified as

$$\hat{u}_i(t) = \hat{u}_i(t-1) + \boldsymbol{K}_i(t)[\boldsymbol{x}^d - \hat{\boldsymbol{x}}(t)] \tag{30}$$

$$\begin{aligned}\boldsymbol{K}_i(t) &= \frac{1}{\lambda(t)}P_i(t-1)\boldsymbol{H}_i(t)^T \\ &\quad \cdot \left[\boldsymbol{I} - \frac{P_i(t-1)\boldsymbol{H}_i(t)\boldsymbol{H}_i(t)^T}{\lambda(t) + P_i(t-1)\boldsymbol{H}_i(t)^T\boldsymbol{H}_i(t)}\right]\end{aligned} \tag{31}$$

$$P_i(t) = P_i(t-1) - \boldsymbol{K}_i(t)\boldsymbol{H}_i(t)P_i(t-1). \tag{32}$$

As covariance matrix $\boldsymbol{R}(t)$ is unknown *a priori*, λ is estimated on-line using the following recursion [20]:

$$\begin{aligned}\hat{\lambda}(t) &= \hat{\lambda}(t-1) + v(t) \\ &\quad \cdot \left[\frac{(\boldsymbol{x}^d - \hat{\boldsymbol{x}}(t))^T(\boldsymbol{x}^d - \hat{\boldsymbol{x}}(t))}{\boldsymbol{n}} - \hat{\lambda}(t-1)\right]\end{aligned} \tag{33}$$

where $v(t) = 1/t$.

This inversion algorithm, as indicated before, is decoupled in nature with respect to input updates. Decoupling of prediction of inputs can help in parallel implementation of the algorithm which is very important advantage for real-time applications.

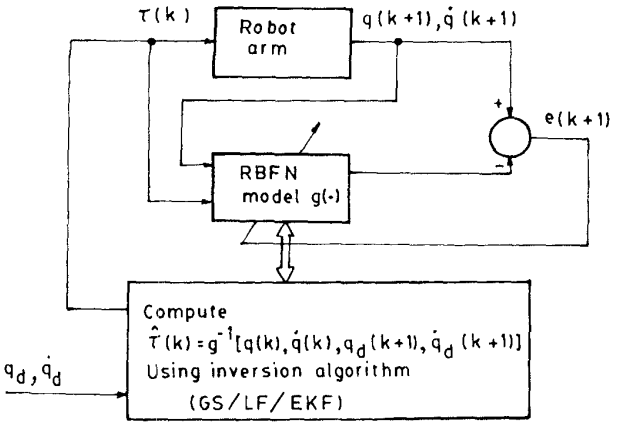


Fig. 2. Proposed controller structure.

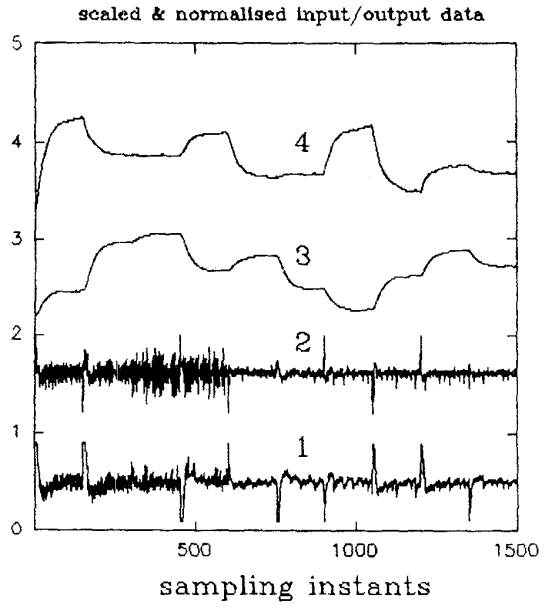


Fig. 3. Training data generated by tracking "pick and place" trajectories: 1) torque input at joint 1; 2) torque input at joint 2; 3) joint 1 position; and 4) joint 2 position. (Input/output data are normalized and scaled differently.)

IV. Neural Controller for Robot Manipulator

The vector equation of motion of the N-link rigid manipulator can be written in the form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\mathrm{q}})\dot{\mathrm{q}} + \boldsymbol{G}(\boldsymbol{q}) = \tau \tag{34}$$

where τ is the $N \times 1$ vector of joint actuator torques, $\boldsymbol{q}$ is the $N \times 1$ vector of joint positions, $\boldsymbol{M}(\boldsymbol{q})$ is an $N \times N$ inertial matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ represent torque arising from centrifugal and Coriolis forces, and $\boldsymbol{G}(\boldsymbol{q})$ represent torques due to gravitational effects.

The model in (34) suffers from two kinds of dynamic uncertainties: structured and unstructured. The structured uncertainties are due to uncertainties in parameter values while unstructured uncertainties are because of unmodeled effects such as friction, joint flexibility, motor backlash, and external

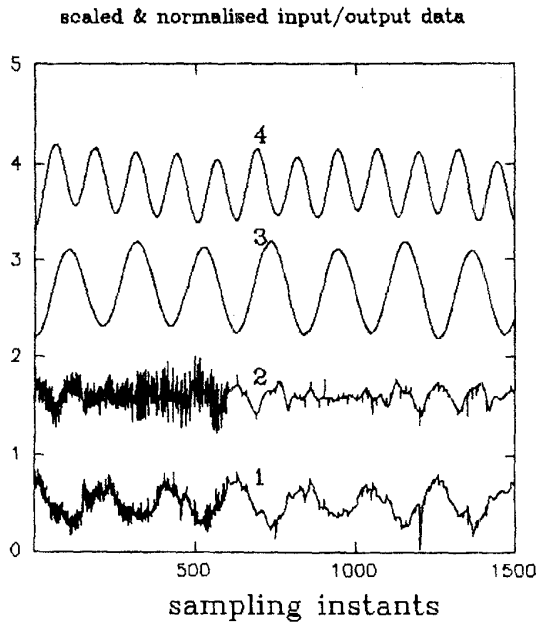


Fig. 4. Training data generated by tracking sinusoid trajectories: 1) torque input at joint 1; 2) torque input at joint 2; 3) joint 1 position; and 4) joint 2 position. (Input/output data are normalized and scaled differently.)

disturbances. To account for both types of uncertainties, we opted for a neural model of the robot arm using *a priori* knowledge of the approximate model (34). Thus in (1), $\boldsymbol{x}(k)$ is chosen as $\boldsymbol{x}(k) = [\boldsymbol{q}(k)^T, \dot{\boldsymbol{q}}(k)^T]^T, 2N \times 1$ vector and $\boldsymbol{u}(k)$ is assigned as $\boldsymbol{u}(k) = \tau(k), N \times 1$ vector.

The RBFN model of the robot arm learned using training data, in the format given in (5), is placed in parallel to the actual plant to take account of on-line dynamic changes. Once the neural emulator of the plant is obtained, the control objective can be defined as follows. Given the desired trajectory $\boldsymbol{x}^d(k+1) = [\boldsymbol{q}_d(k+1)^T, \dot{\boldsymbol{q}}_d(k+1)^T]^T$ and the present state $\boldsymbol{x}(k) = [\boldsymbol{q}(k)^T, \dot{\boldsymbol{q}}(k)^T]^T$, compute the input joint torque $\tau(k)$ using the iterative network inversion algorithm so that RBFN output activation approximates desired response. During inversion, at each sampling instant the initial guess of control input is chosen to be $\tau(k-1)$. The controller structure thus described is shown in Fig. 2.

Prediction of the control input at the Kth sampling instant is carried out in the following steps after neural-network training is over.

1) Get $\boldsymbol{x}(k)$ from sensors, $\boldsymbol{x}^d(k+1)$ from trajectory planner and assign $\hat{\boldsymbol{u}}(k-1)$ to $\hat{\boldsymbol{u}}(k)$.
2) Start iterative inversion; $t = 0$ (iterative step); and $\hat{\boldsymbol{u}}(t) = \hat{\boldsymbol{u}}(k)$

loop: $t = t + 1$
Compute RBFN response $\hat{\boldsymbol{x}}(k+1)$.
If $\|\boldsymbol{x}^d(k+1) - \hat{\boldsymbol{x}}(k+1)\| < \epsilon$, or if $t > t_{\max}$ then
{stop iterative loop and actuate control action
$\hat{\boldsymbol{u}}(k) = \hat{\boldsymbol{u}}(t)$}
else {update input vector $\boldsymbol{u}(k)$ using any of the inversion algorithms
(GS/LF/EKF) and go to *loop*}.

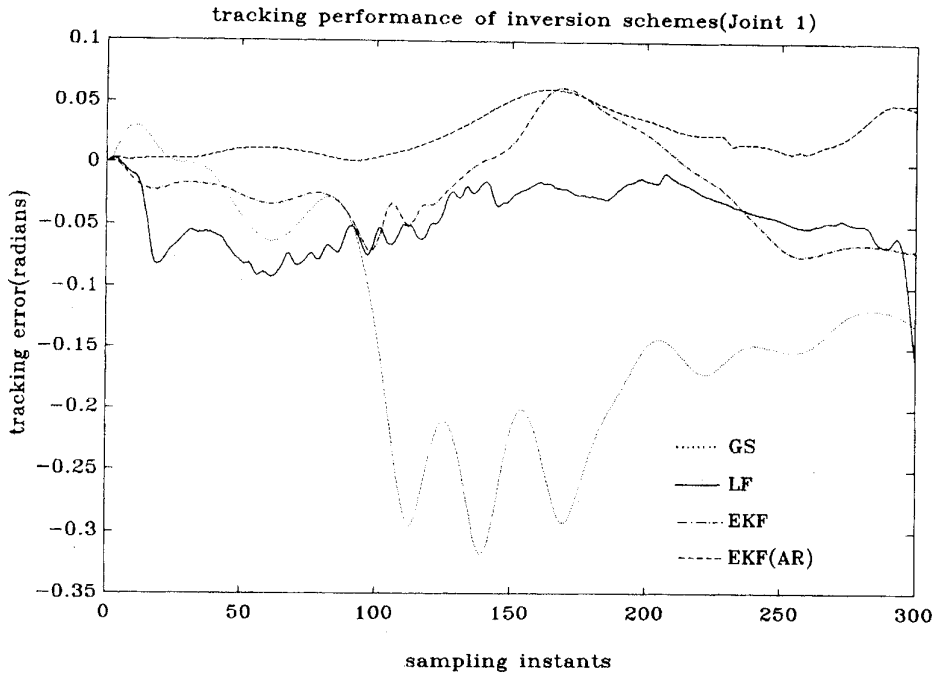


Fig. 5. Position tracking of joint 1 using inversion algorithms GS, LF, EKF, and EKF (after retraining of RBFN model).

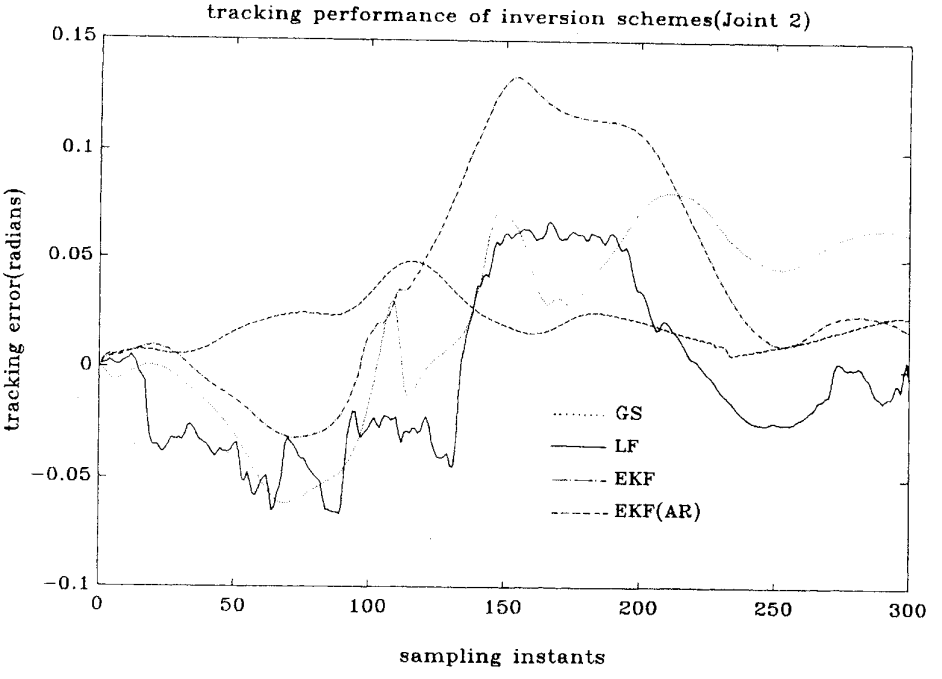


Fig. 6. Position tracking of joint 2 using inversion algorithms GS, LF, EKF, and EKF (after retraining of RBFN model).

V. Simulation Results

As an example, the dynamic equations of the two-degree-of freedom manipulator used by Slotine and Li [1] are used for input–output data generation and controller structure is obtained based on these sets of data only. The robot arm dynamics can be written explicitly as

$$a_1\ddot{q}_1 + (a_3C_{21} + a_4S_{21})\ddot{q}_2 - a_3S_{21}\dot{q}_2^2 + a_4C_{21}\dot{q}_2^2 = \tau_1 \quad (35)$$

$$(a_3C_{21} + a_4S_{21})\ddot{q}_1 + a_2\ddot{q}_2 + a_3S_{21}\dot{q}_1^2 - a_4C_{21}\dot{q}_1^2 = \tau_2 \quad (36)$$

where $C_{21} = \cos(q_2 - q_1), S_{21} = \sin(q_2 - q_1)$. The four parameters a_1, a_2, a_3, and a_4 are taken as 0.15, 0.04, 0.03, and 0.025 kg-m^2, respectively.

A. On-Line Data Generation

The training data to find a neural model of the robot arm are generated using a PD controller to track various random "pick and place" trajectories (Fig. 3) and sinusoid

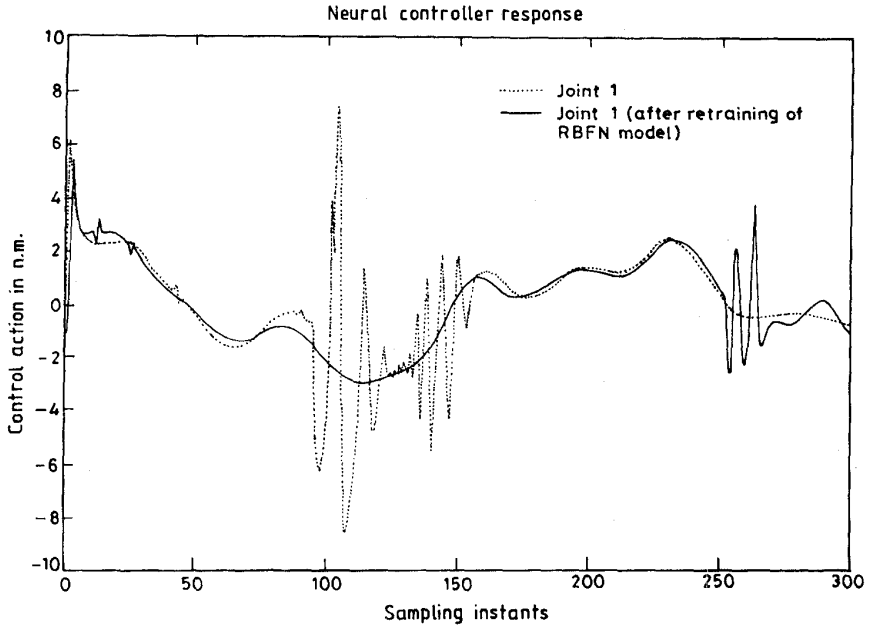


Fig. 7. Neural controller response of joint 1 using EKF based inversion algorithm: before and after retraining of RBFN model.

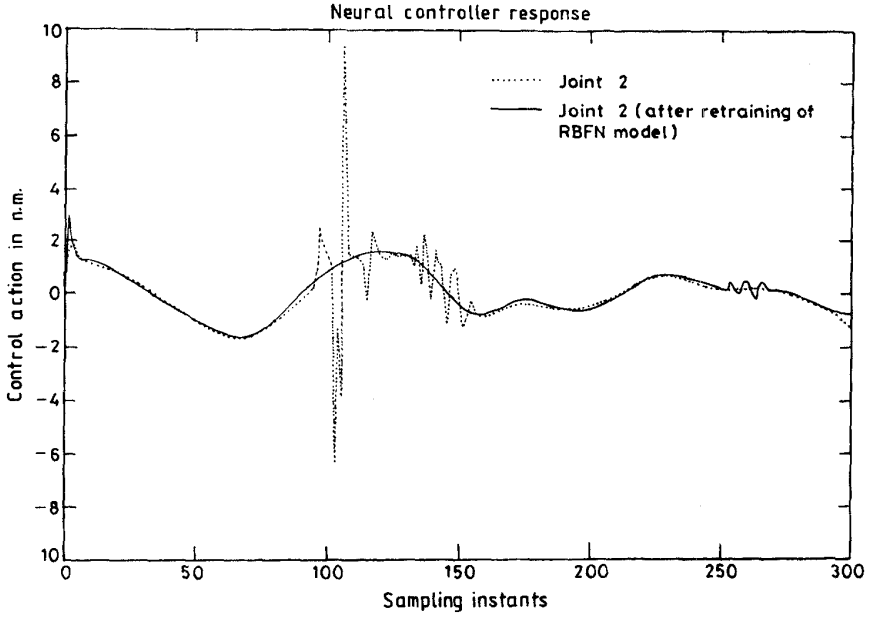


Fig. 8. Neural controller response of joint 2 using EKF based inversion algorithm: before and after retraining of RBFN model.

trajectories (Fig. 4). The robot joint work space is constrained by following boundaries (the joint position is expressed in radian) $0 < q_1 < 5.0, 0 < q_2 < \tau$ and input torque limit is set to $-8.0 < \tau_1, \tau_2 < 8.0$ N · m. While tracking random trajectories at each sampling instant, various dither signals in the form of white noise, impulses, step functions, and ramp and parabolic types of functions are added to the PD controller output to improve generalization capabilities of the RBFN model. In this way we generated 3000 pairs of input–output data taking the sampling interval to be 10 ms. The command sinusoid trajectories for the PD controller are taken as

$$q_d(t) = q_r + Q^{\max}(1 - \cos \omega t).$$

The trajectory parameters $(q^{\max}, \omega)$ expressed in (radian, radian/s) are fixed at values (2.0, 3.0) and (1.2, 5.0) for joint 1 and joint 2, respectively, while q_r is varied randomly over the interval $(0, \pi/4)$.

TABLE I
COMPARATIVE PERFORMANCE OF INVERSION ALGORITHMS POSITION TRACKING OF JOINT 1

Initial Condition $\tau(0)$	Maximum number of iterations t_{max}	rms error in Joint 1 position		
		GS	LF	EKF
-8.0,-8.0	3	0.034	0.02	0.008
	5	0.1	0.08	0.008
	10	0.0469	0.019	0.008
-3.2,4.8	3	0.027	0.007	0.008
	5	0.017	0.008	0.02
	10	0.019	0.016	0.0078
0.0,0.0	3	0.0936	0.038	0.008
	5	0.054	0.0116	0.009
	10	0.033	0.0105	0.01
4.8,-3.2	3	0.061	0.2	0.021
	5	0.39	0.029	0.01
	10	0.08	0.006	0.01
8.0,8.0	3	0.18	0.02	0.018
	5	0.072	0.036	0.01
	10	0.086	0.011	0.01

TABLE II
COMPARATIVE PERFORMANCE OF INVERSION ALGORITHMS POSITION TRACKING OF JOINT 2

Initial Condition $\tau(0)$	Maximum number of iterations t_{max}	rms error in Joint 2 position		
		GS	LF	EKF
-8.0,-8.0	3	0.036	0.1	0.02
	5	0.03	0.38	0.02
	10	0.013	0.057	0.02
-3.2,4.8	3	0.045	0.02	0.02
	5	0.02	0.032	0.018
	10	0.018	0.089	0.019
0.0,0.0	3	0.037	0.09	0.02
	5	0.027	0.0153	0.016
	10	0.015	0.012	0.026
4.8,-3.2	3	0.018	0.43	0.021
	5	0.24	0.044	0.018
	10	0.04	0.018	0.026
8.0,8.0	3	0.1	0.02	0.019
	5	0.035	0.058	0.017
	10	0.045	0.04	0.027

B. RBFN Model

The RBFN model for the two-link manipulator given by (35) and (36) is designed to have six inputs $(\tau(k), \boldsymbol{q}(k), \dot{\boldsymbol{q}}(k))$ and four outputs $(\boldsymbol{q}(k+1), \dot{\boldsymbol{q}}(k+1))$. The model incorporates 100 hidden computing units. Training of the RBFN is carried out using 3000 data pairs for 10 numbers of passes (30 000 iterations). Then following the same technique used for generation of training data, we generated a test data set for 10 new trajectories. After training was over the rms error for the test data set was found to be 0.003.

C. Comparison of Three Inversion Schemes Based on Proposed Controller

The desired trajectories for both joints of the manipulator are chosen as

$$q_{d1}(t) = 1.5(1 - \cos 3t) \tag{37}$$

TABLE III
COMPARATIVE PERFORMANCE OF INVERSION ALGORITHMS VELOCITY TRACKING OF JOINT 1

Initial Condition $\tau(0)$	Maximum number of iterations t_{max}	rms error in Joint 1 velocity		
		GS	LF	EKF
-8.0,-8.0	3	0.093	0.05	0.012
	5	0.085	0.16	0.012
	10	0.043	0.028	0.012
-3.2,4.8	3	0.058	0.035	0.012
	5	0.042	0.032	0.02
	10	0.028	0.04	0.012
0.0,0.0	3	0.092	0.068	0.013
	5	0.054	0.022	0.01
	10	0.034	0.027	0.01
4.8,-3.2	3	0.054	0.26	0.021
	5	0.196	0.034	0.011
	10	0.094	0.017	0.014
8.0,8.0	3	0.13	0.045	0.02
	5	0.077	0.065	0.0168
	10	0.123	0.029	0.015

TABLE IV
COMPARATIVE PERFORMANCE OF INVERSION ALGORITHMS VELOCITY TRACKING OF JOINT 2

Initial Condition $\tau(0)$	Maximum number of iterations t_{max}	rms error in Joint 2 velocity		
		GS	LF	EKF
-8.0,-8.0	3	0.06	0.093	0.011
	5	0.034	0.21	0.01
	10	0.017	0.056	0.0099
-3.2,4.8	3	0.021	0.049	0.01
	5	0.014	0.051	0.01
	10	0.012	0.07	0.01
0.0,0.0	3	0.038	0.088	0.016
	5	0.017	0.038	0.011
	10	0.014	0.029	0.009
4.8,-3.2	3	0.021	0.5	0.018
	5	0.145	0.055	0.011
	10	0.0499	0.031	0.014
8.0,8.0	3	0.09	0.065	0.018
	5	0.029	0.074	0.018
	10	0.06	0.048	0.016

$$q_{d2}(t) = (1 - \cos 5t). \tag{38}$$

The control algorithm presented in Section IV is implemented for all three inversion algorithms. For the gradient search scheme the learning rate η is fixed at 1.0. The Lyapunov function-based inversion scheme, as given in (18) and (20), is implemented choosing μ to be 0.1. The initial values for implementation of the EKF-based inversion algorithm [(30)–(33)] are selected as $P_1 = P_2 = 1.0$ and $\hat{\lambda}(0) = 0.05$. In all the cases, the initial condition for control input at the first sampling instant is chosen as $\tau(0) = [0, 0]^T$ N · m, and the maximum number of iterations per sampling interval is kept at 10. Tracking performance of all the schemes are compared in Figs. 5 and 6 in terms of joint tracking error. Position tracking errors for joint 1 is shown in Fig. 5 while Fig. 6 shows position tracking errors for joint 2. The controller responses for joints 1 and 2 are shown in Figs. 7 and 8, respectively, for the case of the EKF approach. The proposed controller is found to be stable in all the cases though tracking errors are relatively

TABLE V
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER POSITION TRACKING OF JOINT 1

Trajectory Parameters				rms error in Joint 1 position		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.043	0.014	0.026
1.5	1.0	3.0	5.0	0.0336	0.01	0.01
0.5	0.5	1.0	4.0	0.047	0.039	0.047
1.5	1.0	3.0	3.0	0.023	0.014	0.028
1.0	1.0	2.0	3.0	0.034	0.024	0.031

TABLE VI
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER POSITION TRACKING OF JOINT 2

Trajectory Parameters				rms error in Joint 2 position		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.055	0.023	0.06
1.5	1.0	3.0	5.0	0.0156	0.012	0.016
0.5	0.5	1.0	4.0	0.0095	0.022	0.009
1.5	1.0	3.0	3.0	0.06	0.028	0.024
1.0	1.0	2.0	3.0	0.051	0.036	0.044

TABLE VII
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER VELOCITY TRACKING OF JOINT 1

Trajectory Parameters				rms error in Joint 1 velocity		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.038	0.0178	0.045
1.5	1.0	3.0	5.0	0.034	0.027	0.013
0.5	0.5	1.0	4.0	0.034	0.022	0.033
1.5	1.0	3.0	3.0	0.067	0.027	0.029
1.0	1.0	2.0	3.0	0.094	0.028	0.045

large. The error is found to be maximum in the case of GS and minimum in the case of EKF.

Further study was done to evaluate comparative performances of each algorithm by varying the initial condition $\tau(0)$ and allowable number of maximum iteration $t_{\max}$ for the following reasons. Earlier we mentioned that each inversion algorithm computes input $\hat{\tau}(k)$ starting with initial guess $\hat{\tau}(k-1)$ as predicted in the previous sampling instant. But when $k = 1$, the initial condition is not known *a priori*. So we are left with the option of choosing arbitrary an initial condition from the input space. Thus the effect of the initial condition in control input prediction is noteworthy. The second parameter $t_{\max}$ (maximum allowable number of iterations per sampling interval) is important because the control input must be predicted using a fixed number of iterations within the sampling interval.

To study the comparative performance of inversion algorithms, we selected five arbitrary initial conditions as $\tau(0) = [-8.0, -8.0]^T$; $[-3.2, 4.8]^T$; $[0, 0]^T$; $[4.8, -3.2]^T$; and $[8.0, 8.0]^T$, respectively. The parameter $t_{\max}$, maximum number of iterations per sampling interval, is varied at three different values 3, 5, and 10, respectively. The root-mean-square (rms) error over each trajectory is observed while implementing the proposed controller to track the trajectories

TABLE VIII
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER VELOCITY TRACKING OF JOINT 2

Trajectory Parameters				rms error in Joint 2 velocity		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.038	0.028	0.035
1.5	1.0	3.0	5.0	0.014	0.029	0.011
0.5	0.5	1.0	4.0	0.006	0.04	0.006
1.5	1.0	3.0	3.0	0.03	0.042	0.032
1.0	1.0	2.0	3.0	0.035	0.054	0.02

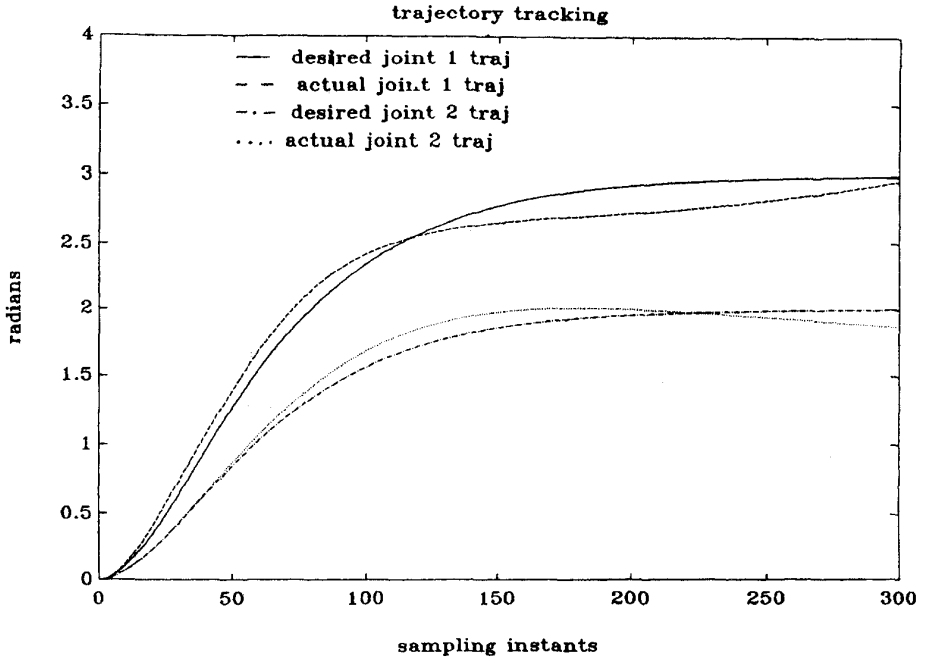


Fig. 9. Trajectory tracking using EKF-based inversion algorithm to see generalization capability of the controller.

given by (37) and (38). The simulation results for GS-, LF-, and EKF-based inversion schemes are compared in Tables I–IV for individual joint position and velocity separately.

The results in these tables show that GS- and LF-based inversion schemes are affected by initial conditions while the performance of EKF-based inverse mapping is practically invariant to these initial conditions. Also, we observe that convergence of the EKF-based inversion scheme is faster because for almost all the cases the algorithm has converged with rms error below 0.021 within the first three iterations, unlike the other two algorithms. But most important of all is the superior tracking accuracy of the proposed controller based on the EKF-based inversion scheme compared to the other two approaches. For example, while tracking the same trajectory for the joint 1 position, Table I shows that the GS approach resulted in a tracking error varying from 0.017 to 0.39, the LF approach resulted in a tracking error varying from 0.006 to 0.2, while for the EKF approach the tracking error shows a small variation from 0.008 to 0.021. These results indicate that rms error per trajectory is minimum in the case of the EKF-based inversion scheme while rms trajectory tracking error is maximum in the case of the GS-based inversion scheme.

The superior performance of the EKF-based inversion approach can be qualitatively explained in the following fashion. The rate of convergence for the gradient search scheme is dependent on a tuning parameter, η, the learning rate, heuristically determined by the user. On the other hand, in EKF, the rate of convergence is dependent on the Kalman gain which is adapted through the iterative process to give the minimum variance estimate of the input activation. Both the GS- and LF-based inversion schemes predict input by minimizing the target error at the emulator output. As the output of the emulator bears a stochastic relationship with the actual response of the robot arm, minimum variance prediction of input is expected to do better, in particular, when the network is partially trained. This explains why the EKF-based

TABLE IX
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER POSITION TRACKING OF JOINT 1 AFTER RETRAINING OF RBFN MODEL

Trajectory Parameters				rms error in Joint 1 position		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.012	0.0046	0.005
1.5	1.0	3.0	5.0	0.008	0.002	0.007
0.5	0.5	1.0	4.0	0.004	0.004	0.0027
1.5	1.0	3.0	3.0	0.009	0.004	0.003
1.0	1.0	2.0	3.0	0.003	0.009	0.007

TABLE X
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER POSITION TRACKING OF JOINT 2 AFTER RETRAINING OF RBFN MODEL

Trajectory Parameters				rms error in Joint 2 position		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.011	0.007	0.013
1.5	1.0	3.0	5.0	0.017	0.007	0.007
0.5	0.5	1.0	4.0	0.004	0.007	0.004
1.5	1.0	3.0	3.0	0.033	0.010	0.013
1.0	1.0	2.0	3.0	0.006	0.013	0.010

TABLE XI
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER VELOCITY TRACKING OF JOINT 1 AFTER RETRAINING OF RBFN MODEL

Trajectory Parameters				rms error in Joint 1 velocity		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.03	0.007	0.0027
1.5	1.0	3.0	5.0	0.004	0.013	0.006
0.5	0.5	1.0	4.0	0.0037	0.010	0.0026
1.5	1.0	3.0	3.0	0.015	0.012	0.006
1.0	1.0	2.0	3.0	0.004	0.0095	0.004

approach performed better in terms of both tracking accuracy and convergence speed.

D. Generalization Capability of the Controller

To study the generalization capability of the proposed controller in tracking any arbitrary trajectory we performed the following simulation. The desired trajectory for each joint is chosen as

$$q_{d1}(t) = q_1^{\max}(1 - \cos\omega_1 t) \tag{39}$$

$$q_{d2}(t) = q_2^{\max}(1 - \cos\omega_2 t). \tag{40}$$

Varying the parameters of the above trajectories such as $q_1^{\max}$, $q_2^{\max}$, ω_1, and ω_2, we observed the tracking performance of the neural controller in terms of rms error considering all three inversion algorithms. Tables V–VIII compare the generalization capability of the controller based on GS-, LF-, and EKF-based inversion schemes, respectively. Considering the fact that RBFN was trained using the data generated by trying to track a single sinusoid trajectory through combination of PD controller output and dither signal, results presented in Tables V–VIII confirm the generalization capability of the RBFN model and the proposed controller. We can conclude that the proposed controller based on each of the inversion algorithms has the capability of generalization since the tracking

TABLE XII
GENERALIZATION CAPABILITY OF PROPOSED CONTROLLER VELOCITY TRACKING OF JOINT 2 AFTER RETRAINING OF RBFN MODEL

Trajectory Parameters				rms error in Joint 2 velocity		
q_1^{max}	q_2^{max}	ω_1	ω_2	GS	LF	EKF
1.0	1.0	2.0	2.0	0.034	0.016	0.0022
1.5	1.0	3.0	5.0	0.0019	0.010	0.004
0.5	0.5	1.0	4.0	0.0076	0.019	0.0018
1.5	1.0	3.0	3.0	0.01	0.021	0.005
1.0	1.0	2.0	3.0	0.002	0.018	0.002

error was relatively small. A further test of generalization is done by making both the joints move from the initial position $\boldsymbol{q}_d = [0, 0]^T$ to the final position $\boldsymbol{q}_d = [3.0, 2.0]^T$ (in radian) following the trajectories given by

$$q_{d1}(t) = 1.5 + 1.5[1. + 6.e^{-t/.3} - 8.e^{t/.4}] \quad (41)$$

$$q_{d2}(t) = 1.0 + [1 + 6.e^{-t/.3} - 8.e^{-t/.4}]. \quad (42)$$

The trajectory tracking shown in Fig. 9 was obtained by implementing the proposed controller using only the EKF-based inversion algorithm. It is verified that the proposed controller is stable while the rms tracking error is 0.021.

Although simulation results by and large confirm the generalization of the controller in tracking any arbitrary trajectory, rms error per trajectory is found to be more than 0.003, the error set for forward learning. Taking account of the fact that RBFN was trained over the training data to achieve rms error 0.003, a relatively large tracking error in trajectory tracking may be attributed to dimensionally insufficient data [18]. To verify this proposition, we conducted further simulations as follows.

E. Generation of Training Examples to Retrain RBFN Model

Our extensive simulation has proved that the proposed controller is robust and stable in tracking any arbitrary trajectory though tracking accuracy is not good. Referring to Figs. 7 and 8, we can also see that the controller response based on the EKF approach is not smooth as expected (generally desired continuous smooth trajectory response is backed by a smooth control action as far as adaptive and nonadaptive schemes are concerned). These facts indicate the partial training of the RBFN model. So we generated new examples (3000 data pairs) tracking 10 more trajectories, represented by (39) and (40) obtained by varying the parameters $q_1^{\max}, q_2^{\max}, \omega_1$, and ω_2 randomly. But here instead of using the PD controller, we implemented the neural controller based on network inversion. Through the inversion process, the network searches in the input space for the appropriate input which would minimize the tracking error when the RBFN model is not completely trained for all possible trajectories in the workspace. The input–output pair so generated can be used to retrain the RBFN model so that it can adapt itself for newer inputs. This learning loop, in effect, provides a self-organizing search strategy for adapting to the complete input space. So as the controller is trying to track the desired trajectories, simultaneously RBFN is trained minimizing the error $E = 1/2\ \Sigma_{i=1}^{4}(x_i - \hat{x}_i)^2$. After the training is over, we repeated the experiment described in the previous subsection. The results are presented in Tables IX–XII. Significant improvement in tracking accuracy verified our presumption of dimensionally insufficient data being the cause of the partially trained RBFN model. The same trajectories given by (37) and (38) were again tracked by the neural controller using the EKF-based inversion scheme, and trajectory tracking is compared with earlier schemes as shown in Figs. 5 and 6. The average rms error is now 0.006 while previously it was 0.0134. The controller response in the present case is compared with the response obtained before retraining of the RBFN model as shown in Figs. 7 and 8. Figs. 7 and 8 show that control action after retraining of the RBFN model is almost smooth. This proves the point that a dimensionally sufficient RBFN model will make the controller more accurate and robust.

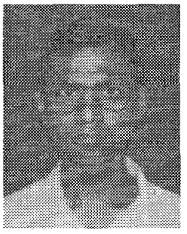
VI. CONCLUSION

We presented a new tracking controller for a robotic manipulator using inversion of its neural emulator. The proposed controller was investigated using three different inversion schemes based on gradient search, Lyapunov function, and EKF approaches, respectively. In all the cases the robustness and stability of the controller are established. Extensive simulation was performed to compare the performance of the three inversion schemes in terms of trajectory tracking error and convergence speed. It is shown that the EKF-based inversion scheme is better compared to LF- and GS-based inversion algorithms.

REFERENCES

[1] J.-J. E. Slotine and W. Li, "Composite adaptive control of robot manipulators," *Automatica*, vol. 25, no. 4, pp. 509–519, 1989.
[2] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York: Wiley, 1989.
[3] M. Vukobratovc, D. Stokic, and N. Kircanski, *Nonadaptive and Adaptive Control of Manipulation Robots*. New York: Springer-Verlag, 1985.
[4] J. Craig, *Adaptive Control of Mechanical Manipulators*. Reading, MA: Addison-Wesley, 1988.
[5] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 356–366, 1989.

[6] M. J. D. Powell, "Radial basis function approximations to polynomials," in *Proc. 12th Biennial Numerical Analysis Conf.*, 1987, pp. 223–241.
[7] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamic systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.
[8] W. T. Miller, F. H. Glanz, and L. G. Kraft, III, "Application of a general learning algorithm to the control of robotic manipulators," *Int. J. Robot. Res.*, pp. 84–98, 1987.
[9] K. Y. Goldberg and B. A. Pearlmutter, "Using backpropagation with temporal windows to learn the dynamics of the CMU direct drive arm II," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989.
[10] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback error learning neural networks for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, pp. 251–265, 1988.
[11] D. Bassi and G. Bekey, "High precision position control by Cartesian trajectory feedback and connectionist inverse dynamics feedforward," in *Proc. Int. Joint Conf. Neural Networks*, Washington, D.C., 1989, pp. 325–332.
[12] H. Gomi and M. Kawato, "Neural-network control for a closed loop system using feedback error learning," *Neural Networks*, vol. 6, pp. 933–946, 1993.
[13] M. I. Joradan, "Supervised learning and systems with excess degree of freedom," Univ. Massachusetts, Amherst, COINS Tech. Rep. 88-27.
[14] A. Y. Zomaya and T. M. Nabhan, "Centralised and decentralised neuro-adaptive robot controllers," *Neural Networks*, vol. 6, pp. 223–244, 1993.
[15] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavior science," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.
[16] D. Nguyen and B. Widrow, "Neural networks for self learning control systems," *Int. J. Contr.*, vol. 54, no. 6, pp. 1439–1451, 1991.
[17] A. Linden and J. Kindermann, "Inversion of multilayer nets," in *IJCNN*, Washington, D.C., June 1989, pp. II 425–II 430.
[18] D. A. Hoskins, J. N. Hwang, and J. Vagners, "Iterative inversion of neural networks and its application to adaptive control," *IEEE Trans. Neural Networks*, vol. 3, pp. 292–301, 1992.
[19] S. Lee, "Supervised learning with Gaussian potential," in *Neural Networks for Signal Processing*, B. Kosko, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[20] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real time learning algorithm for a multilayered neural network based on extended Kalman filter," *IEEE Trans. Signal Processing*, vol. 40, Apr. 1992.
[21] R. S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks," *IEEE Trans. Signal Processing*, vol. 40, Jan. 1992.
[22] S. Chen, S. A. Billings, and P. M. Grant, "Nonlinear system identification using neural networks," *Int. J. Contr.*, vol. 51, no. 6, pp. 1191–1214, 1990.
[23] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Networks*, vol. 2, pp. 302–309, Mar. 1991.
[24] T. A. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, Sept. 1990.
[25] P. E. An, M. Brown, and C. J. Harris, "Comparative aspects of neural network algorithms for on-line modeling of dynamic processes," in *Proc. Inst. Mech. Eng., Part I*, vol. 207, 1993, pp. 223–241.
[26] S. Chen, S. A. Billings, and P. M. Grant, "Recursive hybrid algorithm for nonlinear system identification using radial basis function networks," *Int. J. Contr.*, vol. 55, no. 5, pp. 1051–1070, 1992.
[27] C.-L. Chen, W.-C. Chen, and F.-Y. Chang, "Hybrid learning algorithm for Gaussian potential function networks," *IEE Proc.-D*, vol. 140, no. 6, Nov. 1993.
[28] J. Moody and C. Darken, "Fast learning in networks for locally tuned processing units," *Neural Computa.*, vol. 1, pp. 281–294, 1989.
[29] A. H. Jazwinski, *Stochastic Process and Filtering Theory*. New York: Academic, 1970.



Laxmidhar Behera was born in January 1967. He received the bachelor's degree and master's degree in electrical engineering from the Regional Engineering College (REC), Rourkela, India, in 1988 and 1990, respectively. He carried out his doctoral research at the Indian Institute of Technology, Delhi, from 1991 to 1995.

He worked as a Lecturer at the REC from 1990 to 1991. At present, he is working as an Assistant Professor at the Birla Institute of Technology and Science, Pilani. He is now actively associated with the research work carried out at the Centre for Robotics, BITS, Pilani. His areas of interests are intelligent robotics, neural networks, adaptive fuzzy control, and evolutionary computation.

Madan Gopal is a Professor in the Department of Electrical Engineering at the Indian Institute of Technology, Delhi. He has approximately 60 research publications to his credit. He has guided many research projects leading to the award of Ph.D. degrees. His current research interests are in the areas of robotics, neural networks, and fuzzy control. He is the author of four books in control engineering. He is the author of a video course in control engineering, distributed through the Foundation for Innovation and Technology Transfer, Delhi. Dr. Gopal is a Fellow of the Institution of Engineers, India.

Santanu Chaudhury received the B.Tech. degree in electronics and electrical communication engineering and the Ph.D. in degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1984 and 1989, respectively.

He is currently an Associate Professor in the Department of Electrical Engineering at the Indian Institute of Technology, Delhi. His research interests are neural networks, pattern recognition, and computer vision.

Dr. Chaudhury was a recipient of the INSA (Indian National Science Academy) Young Scientist's Medal in 1993.