Abstract: The paper is concerned with the design of a hybrid controller structure, consisting of the adaptive control law and a neural-network-based learning scheme for adaptation of time-varying controller parameters. The target error vector for weight adaptation of the neural networks is derived using the Lyapunov-function approach. The global stability of the closed-loop feedback system is guaranteed, provided the structure of the robot-manipulator dynamics model is exact. Generalisation of the controller over the desired trajectory space has been established using an online weight-learning scheme. Model learning, using a priori knowledge of a robot arm model, has been shown to improve tracking accuracy. The proposed control scheme has been implemented using both MLN and RBF networks. Faster convergence, better generalisation and superior tracking accuracy have been achieved in the case of the RBF network.

1 Introduction

In the literature of neural control, as applied to robot tracking problems, the primary focus has been given to various schemes of off-line and on-line inverse dynamics learning using neural networks [14]. In this paper we have proposed a hybrid controller scheme, combining positive aspects of both the classical adaptive control [S-16] and the neural modelling [11-16]. Adaptive control algorithms make use of properties of robot dynamics such as linearity, skew-symmetry etc., on the basis of a priori knowledge of the robot model. On the other hand, inverse dynamics learning, using neural networks, does not make use of well-established knowledge of the rigid robot model. Combination of the adaptive tracking principle, based on a valid model, and neural modelling of robot model components subjected to uncertainties, is expected to provide an efficient performance-oriented controller.

Ozaki et al. [17] have proposed a controller in an adaptive computed torque control configuration, where a neural network has been used to learn the elements of the inertial and Coriolis force matrices and the gravity force vector. It has been shown that this technique has the capability to compensate unmodelled effects, such as friction etc. However, convergence of the weight-learning scheme in the control loop has not been established. Kuan, Whittle and Bavarian [18] have also proposed an adaptive controller, which uses single layer adaline-like neurons for learning the structured uncertainties of a three-degrees-of-freedom robot model, using the Lyapunov-function approach for the weight adaptation.

Motivated by these results, we have presented, in this paper, a new hybrid controller consisting of adaptive control law and a neural estimator. The control law has been adopted from the adaptive algorithm proposed by Slotine and Li [SI. The robot model has been linearly parametrised in terms of inertial, Coriolis and gravity force matrices and/or vector elements. These time-varying parameters have been estimated using feedforward neural networks as functions of joint position and/or velocity. Convergence of the weight-learning scheme has been established using the Lyapunov-function approach. The proposed learning algorithm estimates the manipulator parameters, while the learning algorithm of Kuan et al. [18] only compensates for the uncertainties found in the available robot model. Also, the present scheme uses more powerful MLN and RBFN as neural estimators, instead of simple adalines as in [18].

Two approaches have been followed to train the neural estimator. In one approach, the weights are tuned using an on-line adaptation algorithm only. In the second approach the a priori model knowledge has been utilised to train the network off-line. Subsequently, on-line adaptation has been carried out to compensate for parameter uncertainties and unmodelled dynamics.

2 Parameter estimation using feedforward networks

Neural networks can be used for estimating parameters of an adaptive control algorithm designed for a robot manipulator. Such parameters are nonlinear functions of time-varying joint position and velocity vectors. Both MLN [19] and RBFN [20] are capable of approximating any nonlinear mapping and, hence, can be used as estimators.

In case of a multi-layered network, the standard backpropagation algorithm is very effective, in spite of its limitations such as slow convergence and local
always valid. As robot dynamics is complex in structure, calculation of \( M, C \) and \( G \) on the basis of these slowly time varying. These assumptions may not be payload etc., by assuming them to be either constant or metrised adaptation algorithm [SI, M, C and \( G \) are position and the velocity trackjng errors converge to convergent for the model in eqn. 1 when \( M(q) = M(q) \), \( C(q, q) = C(q, q) \), and \( G(q) = G(q) \), and both the

The adaptive control law is so designed that feedback control operates in a stable mode, provided the exact structure of the manipulator dynamics model is known a priori. The proposed control scheme has been designed based on this concept. The controller structure is shown in Fig. 1. The neural estimator tracks the time-varying controller parameters subjected to uncertainties. Final convergence of the scheme can be established, by ensuring the convergence of the weight learning scheme and global convergence of the closed-loop adaptive system. We now present the control and parameter adaptation law.

![Fig. 1 Neuro-adaptive controller structure](image)

Consider an \( N \)-link rigid robot model given by

\[
M(q)\dot{q} + (3, 44 + G(s))q = T.
\]

where \( z \) is the \( N \times 1 \) vector of joint torques, \( q \) is the \( N \times 1 \) vector of generalised joint positions, \( M(q) \) is an \( N \times N \) inertial matrix, \( C(q, q) \) represents torques arising from centrifugal and Coriolis forces and \( G(q) \) represents torques due to gravitational effects when the manipulator is moving in its workspace.

We have chosen the control law, given by Slotine and Lee [SI]

\[
T = \dot{M}(q)\dot{q} + \dot{C}(q, q)\dot{q} + G(q) - K_u\dot{q}.
\]

where \( \dot{q}, \dot{q} = \dot{q}_l - \dot{q}_r, \dot{q} = \dot{q}_l - \dot{q}_r = \int Rq \dot{q} \) and \( A \) is the gain matrix. The control law in eqn. 2 is globally convergent for the model in eqn. 1 when \( M(q) = M(q), C(q, q) = C(q, q), \) and \( G(q) = G(q), \) and both the position and the velocity tracking errors converge to the sliding surface given by \( s = q - \dot{q} \). In a parametrised adaptation algorithm [SI, M, C and \( G \) are computed on the basis of estimates of unknown manipulator parameters such as link length, link mass, payload etc., by assuming them to be either constant or slowly time varying. These assumptions may not be always valid. As robot dynamics is complex in structure, calculation of \( M, C \) and \( G \) on the basis of these parameters is also computationally expensive. Therefore, instead of estimating manipulator parameters, if we can directly estimate elements of these matrices, the computational cost can be reduced. Use of neural networks as estimators for these matrix elements provides suitable means for handling time-varying uncertainties associated with the robot model.

In our proposed hybrid controller, we estimate \( M, C \) and \( G \) directly using a feedforward neural network. Hence no restrictive assumptions regarding the nature of the controller parameters are required. Since elements of function matrices are being estimated directly, computational complexity is less. We now derive the adaptation law and corresponding weight-learning scheme.

### 3.1 Adaptation law for a neural estimator

For an \( N \)-link robot manipulator, we define the parameter vector \( a \) to be consisting of all elements of inertial, Coriolis and centrifugal, and gravitational force matrices and/or vectors. This \( p \)-dimensional vector \((p = N(2N + 1))\) can be expressed as

\[
a = [M_1, M_2, \ldots, M_{NN}, C_1, C_1, \ldots, C_{NN}, G_1, G_1, \ldots, G_{NN}].
\]

Each element of this parameter vector \( a \) is time varying. These time-varying elements can be estimated using three feedforward neural networks, as shown in Fig. 2. Inertial and gravitational elements can be modelled as functions of a joint position vector while Coriolis and centrifugal terms can be modelled as functions of both the joint position and velocity vectors. During the on-line adaptation process, these network weight vectors are tuned so that tracking convergence is achieved. As outputs of these supervised networks, i.e. the target vectors are unknown, we need to estimate either the actual target vector \( a \) or the error \( e = a - \dot{a} = -\dot{a} \) at the target. Now we will provide a globally convergent scheme to estimate the target error \( e \) on-line, based on output tracking error.

![Fig. 2 Structure of neural estimators (NN, NN, and NN)](image)

Using the linear parametrised property, robot dynamics, eqn. 1, can be written as

\[
Y_T(q, q, q)|a - r.
\]

where

\[
Y_T = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \
\end{bmatrix}
\]
Now, the Lyapunov function can be defined as

\[ V = \frac{1}{2} s^T M s + i a^T \hat{r} a \]  

(6)

where \( F \) is strictly Hurwitz.

The time derivative of the Lyapunov function can be given as

\[
\dot{V} = s^T \dot{S} + a^T \dot{r} + \frac{1}{2} s^T \dot{M} s \\
= s^T (-c s - G s - \dot{r}) + a^T \dot{i} s + s^T \left[ \frac{1}{2} (H \hat{r} - 2C) + C \right] s
\]

(7)

Using the skew symmetric property and control law given in eqn. 2, we have

\[
\dot{V} = s^T (M \dot{q} + C \dot{q} + G - K s) + 5 \dot{r} a
\]

Using the linearity relation, eqn. 4, associated with the robot dynamics model, eqn. 1, we can write

\[
\dot{M}(q) \dot{q} + \dot{C}(q, 4) \dot{q} + G(q) = Y(q, s, \dot{s}, \ddot{s}, \dddot{s})
\]

(8)

where \( U(\cdot) \) has the same function form as that of \( Y(\cdot) \), given in eqn. 4. Thus we have

\[
\dot{V} = s^T (Y \dot{6} - K s) + i \dot{T}^6
\]

(9)

by choosing the adaptation law as

\[
\dot{4} = -r + \dot{Y}^T (q, 4, 4, 7 \hat{r}) s
\]

(10)

Thus the target-error vector, estimated using the adaptation law, eqn. 9, will result in global convergence of the closed-loop adaptive system. The recursive form of the target-error (e) vector estimation can be written as

\[
\hat{a} = -\Gamma^{-1} \dot{Y}^T (q, \dot{q}, \ddot{q} - q_l) s
\]

(11)

where \( p \) is the update rate. Thus, for each desired trajectory pattern, the error at the target of the neural estimator can be computed using eqn. 11. In an iterative fashion. This error now can be backpropagated to update the weights of the neural estimator, using a suitable weight-adaptation algorithm.

As in the case of the parametrised adaptive scheme, the parameter vector consists of constant but unknown parameters. The parametrised adaptation algorithm can be derived from eqn. 9 as follows:

\[
\hat{a} = -\Gamma^{-1} \dot{Y}^T (q, \dot{q}, \ddot{q} - q_l) s
\]

Before the proposed hybrid controller can be used for on-line trajectory tracking, the neural estimator should be trained properly. This training can be done in two ways.

### 3.2 On-line adaptive learning

In this approach, the neural weights are initialised randomly. Then the robot arm is made to track various random trajectories in its workspace. For each desired trajectory pattern, the target error for the neural estimator is computed using eqn. 11. The target error is backpropagated through the network estimator to update its weights. This training process is continued for various desired trajectory patterns spanning the robot workspace. Once training is over, the neural estimator can be directly used to predict the controller parameters, without further update of its weights, provided there are no on-line dynamic uncertainties and disturbances. In case that the robot environment is not free from on-line uncertainties, an on-line adaptation scheme has to be operated, while the manipulator is tracking new trajectories for performing various tasks.

### 3.3 Hybrid learning

In hybrid learning, the training process is carried out in two phases: model learning and an on-line adaptation algorithm. In the first phase, neural estimators are directly trained using the available model. For example, \( M(q) \) element of the inertial matrix, can be modelled as a function of the joint position vector, using the data generated from the known functional available in the model. Thus, through model learning, the controller can be neurally expressed in terms of available knowledge of the robot model before we can start with the on-line adaptation scheme. On-line learning is carried out to take care of various uncertainties. The objective of the hybrid learning is to effectively reduce the duration of on-line learning and to increase the accuracy.

### 4 Simulation

Since nonlinear intricacies of a robot arm are largely reflected in the 2nd and 3rd links, we have considered a two-link manipulator model of PUMA 560 [24], consisting of 2nd and 3rd arms for simulation. The robot dynamics can be explicitly expressed as:

\[
\begin{array}{c}
\frac{a_1 + 0.1 \cos 2 \theta_1}{2} + \frac{a_2}{2} + a_3 \cos 41 + a_5 \cos (q_1 + q_2) = r_1 \\
\frac{a_1 + a_2 \cos (71 \hat{q}_1 + a_1 \hat{q}_2)}{2} - \frac{a_3 \sin q_2 + a_4 \cos 41 + a_5 \cos (q_1 + q_2)}{2} = r_2
\end{array}
\]

(13)

where \( a_1 = 3.82, a_2 = 2.12, a_3 = 0.71, a_4 = 81.82 \) and \( a_5 = 24.06 \).

The objective of simulation is to evaluate the performance of the neuro-adaptive scheme in comparison to the classical parametrised adaptive algorithm. Simultaneously, the performance of the MLN based neural estimator has been compared with a RBF network-based neural estimator.

While doing simulation for the classical parametrised adaptive control scheme, we have assumed that eqns. 13 and 14 represent a true model of the manipulator, and the parameter vector is assumed to be unknown.

While doing simulation for the neuro-adaptive control scheme, the parameter vector is selected based on eqns. 13 and 14, viscous and Coulomb friction forces have been introduced into the true model. Thus, the true model is assumed as follows:

\[
M(q) \ddot{q} + C(q, 4) \dot{q} + G(q) = Y(q, s, \dot{s}, \ddot{s}) + \hat{F}
\]

(15)

where

\[
V = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}
\]

and

\[
F = \begin{bmatrix} k_1 \operatorname{sgn}(q_1) \\ k_2 \operatorname{sgn}(q_2) \end{bmatrix}
\]

\( k_1 = k_2 = 1 \)

We evaluate the performance of the proposed controller, with respect to the true model, so that the capability of the controller to compensate for unmodelled dynamics can be assessed.
The control law $2 = Y(q, \dot{q}, \ddot{q}, a_0)$ - Kos primarily involves the computation of matrix $Y$. For classical parametrisation adaptive control, with the choice of the parameter vector as $a = \{a_1, a_2, a_3, a_4, a_5\}$, the elements of matrix $Y$ can be written as:

$$Y_{11} = \dot{q}_1$$

$$Y_{12} = 9r_1 \cos q_1 + \frac{1}{2} \cos q_2 + \frac{1}{2} r_2 \cos q_1 + \frac{1}{2} \cos q_2$$

$$Y_{13} = \frac{1}{2} \cos q_2$$

$$Y_{14} = \cos q_1, Y_{15} = \cos q_1$$

$$Y_{21} = 0$$

$$Y_{22} = 2 \cos q_2, Y_{23} = \cos q_1 + \frac{1}{2} \cos q_2$$

$$Y_{24} = \cos q_1, Y_{25} = \cos q_1$$

For the proposed control structure, the parameter vector is chosen as $a = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}\}$. With this choice of parameter vector, the matrix $Y$ can be expressed as:

$$Y = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Comparing eqns. 16 and 17, we can observe the simple structure of matrix $Y$ for the case of the hybrid scheme. As both the controller structure and the parameter adaptation algorithm involve this matrix $Y$, computational complexity has been reduced considerably for the case of the proposed hybrid scheme.

### 4.1 Architecture of neural estimators

Three neural networks (NN, NN-1 and NN-2) are used to estimate the parameter vector (Fig. 2). Based on a priori knowledge of the model, the elements of the neural matrix are modelled as functions of $q_1$ (NN-1), joint 2 position only, the elements of the Coriolis and centrifugal force matrix are modelled as functions of all the joint states (NN-2), while elements of the gravity force vector have been modelled as functions of joint 1 and 2 positions (NN-3). In the case of MLN, the structures of NN-1, NN-2 and NN-3 are 2-6-4, 5-6-4 and 3-20-2, and the output layer is chosen to be linear. Each of the input vectors for all these networks contains a bias input. For the RBF network, each of them can be represented as (eqn. 12). In all the cases, the gain matrices are assumed to be $A = 101$ and $K = 201$. Results have been compiled for four trained trajectories in Table 1 and for four test (new) trajectories in Table 2. The corresponding tracking performance for selected trajectories are shown in Figs. 3 and 4, respectively.

### 4.2 Tracking performance using only on-line adaptation

We selected 20 random sinusoid trajectories of varied frequency and amplitudes in robot workspace. Each trajectory is described by the following equation:

$$q^d = \frac{a_0 \cos(\omega t)}{2}$$

Each of them can be represented as $(q_{m_{\theta}}^{\text{data}}, \omega)$. Using the on-line adaptation algorithm, we trained the neural estimator using both MLN and RBFN. For MLN, we trained the network in 600,000 learning steps and RBFN has been trained using the same set of data pairs in 300,000 learning steps. After training, we tested the neural estimator on both the training set and test trajectories. Performance of the hybrid scheme has been compared with the parametrised adaptive scheme (eqn. 12). In all the cases, the gain matrices are assumed to be $A = 101$ and $K = 201$. Results have been compiled for four trained trajectories in Table 1 and for four test (new) trajectories in Table 2. The corresponding tracking performance for selected trajectories are shown in Figs. 3 and 4, respectively.

### 4.3 Tracking performance using hybrid learning

Here we trained the neural estimators, using a priori knowledge of the robot dynamics model. For simulation, we assumed that the available manipulator parameters are subjected to 40-60 % uncertainties over the true robot manipulators given by eqns. 13 and 14. Also we assumed that viscous and friction terms are not available. Training data were generated using the available model. After model learning was over, on-line adaptation was carried out to compensate for both the parameter uncertainties and unmodelled effects, such as viscous and Coulomb friction. For on-line adaptation, for MLN we performed 240 000 learning steps, while for RBFN we performed 120 000 learning steps. After training was over, we evaluated the controller performance by tracking four trajectories from the training set and four test (new) trajectories, as done in the previous sub-section. The corresponding tracking performance in terms of rms error is summarised in Tables 3 and 4.

<table>
<thead>
<tr>
<th>Table 1: Tracking performance on learned trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trajectory parameters for joints 1 and 2 (q_{\text{NN}}, \theta)</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Adaptive</td>
</tr>
<tr>
<td>(1.26), (2.4)</td>
</tr>
<tr>
<td>(3.5), (23)</td>
</tr>
<tr>
<td>(1.63), (3.5)</td>
</tr>
<tr>
<td>(2.4), (1.66)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Tracking Performance on test trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trajectory parameters for joints 1 and 2 (q_{\text{NN}}, \theta)</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>Adaptive</td>
</tr>
<tr>
<td>(2.5), (23)</td>
</tr>
<tr>
<td>(2.43), (1.65)</td>
</tr>
<tr>
<td>(3.4), (1.66)</td>
</tr>
<tr>
<td>(1.66), (3.4)</td>
</tr>
</tbody>
</table>

4.4 Summary of simulation results

From Tables 1 and 2, and Figs. 3-10, we can see that the neuro-adaptive scheme has performed much better in terms of tracking accuracy, in comparison to the classical parametrised adaptive scheme, in spite of the presence of unmodelled dynamics, such as viscous and Coulomb friction forces in the former case. The control scheme, using an RBFN-based neural estimator, has achieved better tracking accuracy than the MLN-based estimation scheme. Generalisation performance of RBFN is much better compared to MLN, as far as tracking errors over test trajectories are concerned. Also we can see, in some cases MLN performance is comparable with that of classical adaptive scheme. In terms of learning speed, RBFN outperforms MLN, as shown in Table 5.

In the case of hybrid training, it has been found that, after model learning, learning steps involved during online adaptation have been drastically reduced. Simultaneously significant improvement in the tracking accuracy has been also observed.

Table 5: Learning speed

<table>
<thead>
<tr>
<th>Training scheme</th>
<th>MLN</th>
<th>RBFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online training</td>
<td>600 000</td>
<td>300 000</td>
</tr>
<tr>
<td>Hybrid training</td>
<td>240 000</td>
<td>120 000</td>
</tr>
</tbody>
</table>

5 Conclusion

We have presented a neuro-adaptive hybrid control scheme, where knowledge of a robot dynamics model has been properly utilised to obtain a high-precision and computationally less intensive control scheme. Simulation results show that the proposed scheme is better than the classical parametrised adaptive scheme in providing very accurate tracking. Through simulation, it has also been established that the RBFN-based neural estimator outperforms its MLN counterpart in achieving better accuracy and generalisation.

6 References

1. GOLDBERG, K.Y., and PEARLMUTTER, B.A.: 'Using back-propagation with temporal windows to learn the dynamics of the CMU direct drive arm 11 in TOURETSKY, D.S. (Ed.): ‘Advances in neural inf. processing systems’ (Morgan Kaufmann, 1989)
3. JORDAN, M.I.: 'Supervised learning and systems with excess of degrees of freedom', CDINS Tech. Rep., No. 88-27, University of Massachusetts at Amherst