Performance of a neuro-model-based robot controller: adaptability and noise rejection

by A. N. Poo, M. H. Ang Jr., C. L. Teo and Qing Li

Effective control strategies for robotic manipulators usually require the *on-line* computation of the robot dynamic model in real time. However, the complexity of the robot dynamic model makes this difficult to achieve in practice, and multiprocessor controller architectures appear attractive for real-time implementation inside the control servo loop. Furthermore, inevitable modelling errors, changing parameter values and disturbances can compromise controller stability and performance. In this paper, the performance of a neuro-model-based controller architecture is investigated. The neural network is used to adapt to unmodelled dynamics and parameter modelling errors. Simulation of the neuro-model-based control of a one-link robot demonstrates an improved performance over standard model-based control algorithm, in the presence of modelling errors and in the presence of disturbance and noise.

1 Introduction

Robotic manipulators are programmed to interact with the environment by positioning and orienting an end-effector in space. The robot motion control problem centres around the design of a stable and robust algorithm to co-ordinate the joint motions and enable the robot to follow a specified trajectory in world (Cartesian) co-ordinates. Since the robot joints define the axes along (or about) which motion can occur, the controller has to resolve the motion of the end-effector into joint co-ordinates [1]. The motion control problem focuses on the computation of the actuating joint forces/torques that are required to produce the desired position and orientation trajectory of the end-effector.

The equations of motion of a manipulator are typically very complex, highly coupled and highly non-linear. However, most commercial manipulators are equipped with controllers that ignore these non-linear robot dynamics. These simplified controllers fail to characterise the complex joint dynamics and coupling, resulting in oscillations or overshoot of the end-effector. The continuously increasing demands for productivity and precision have imposed special requirements on the robot controller, and caused a shift of emphasis towards more sophisticated controllers, the design of which includes consideration of the dynamics of robotic manipulators. This has led to the development of *model-based control* (also referred to as *non-linear feedback control*) algorithms [2] to achieve better performance and take into account

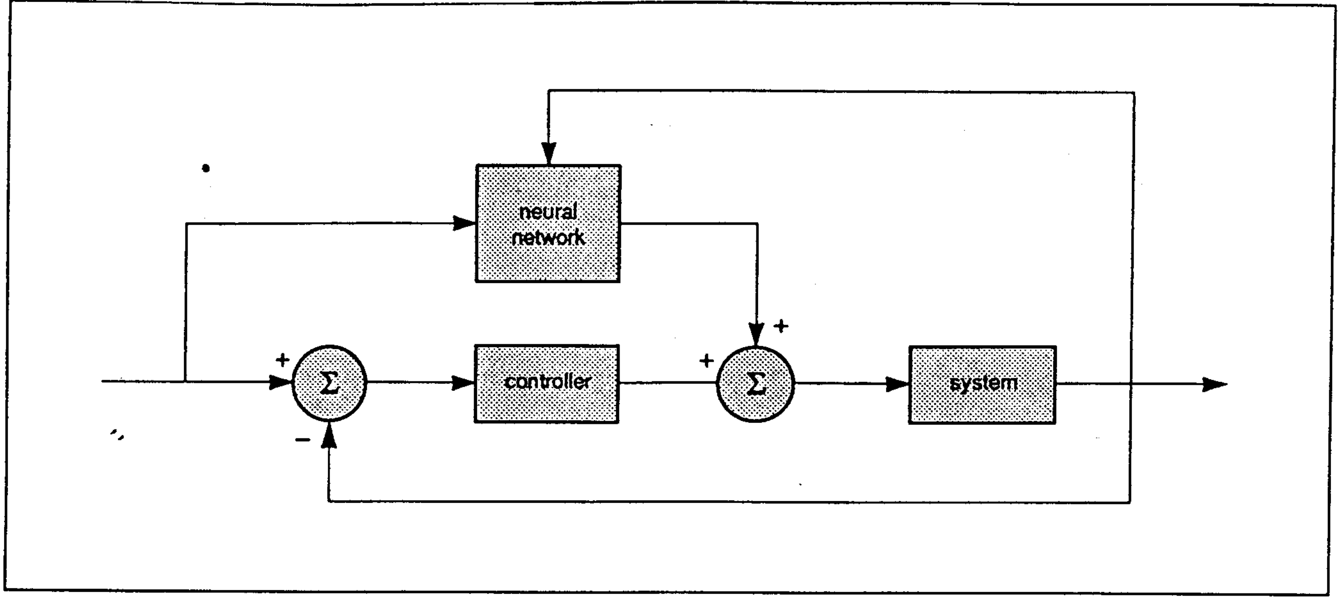

Fig. 1 Neural network as a feedforward controller

the dynamics of the robot.

There are two barriers to the successful implementation of model-based control.

- The computation of the complex non-linear dynamic model of the robot needs to be done in real-time. For effective control, this should typically be less than 10 ms. This is hard to achieve with present single-processor technology, and specialised (or customised) multiprocessor architectures may be needed for possible real-time implementation inside the control servo loop.
- The parameters in the dynamic model of the robot must be known precisely. This is because model-based control for robots is *not* robust in the presence of *modelling*, *parameter* errors and *disturbances*. Modelling errors are introduced by unmodelled dynamics (e.g. joint flexibility) and/or simplified models that are designed to reduce the real-time computational requirements of the controller. Parameter errors arise from practical limitations in the knowledge of numerical values for the kinematic and dynamic parameters of the robot and/or from pay-load variations. Disturbances such as load disturbances and measurement noise are inevitable in any control system and can severely degrade a system's performance.

Adaptive control algorithms proposed in the robotics literature to handle parameter variations and unmodelled dynamics have achieved some success in controlling a robot. However, adaptive control theory [3] has been developed only for linear systems with constant unknown parameters; although in practice, it works for systems with slowly time-varying unknown parameters. Current research efforts focus on adaptive algorithms that learn and update the robot dynamic parameters, as well as those that compensate for modelling errors. One such method uses *predictors* to approximate and compensate for modelling errors [4]. Another interesting approach is the *α-computed-torque* non-linear feedback control algorithm, which is designed to ameliorate modelling problems [5]. Practical implementation of adaptive control algorithms is, however, difficult because of the amount of computation and short sampling times required.

In this paper, we present a novel adaptive control algorithm based on the model-based control approach, but using a neural network to 'learn' the model. The parallel nature of neural networks means that it can be significantly faster than conventional controllers when implemented in a multi-processor architecture system. The neural network was trained using simulated *clean* data based on a model and using data corrupted with noise to simulate *on-line noisy* data. In this paper, we refer to the latter as *NET*2 and the former as *NET*1. Once such a neural-network controller is trained, it can then be used to replace the conventional controller. As is demonstrated later, this neuro-model-based controller does not suffer from the aforementioned limitations of model-based and adaptive control algorithms. The neural network is able to learn and adapt to parameter variations and unmodelled dynamics. Simulation results also show that the neural network controller can track a desired trajectory even in the presence of measurement noise and disturbances.

2 Neural networks for robot control

In the robotics community, there is currently a renewed and growing interest in using *neural network* technology [6]. The neural-network architecture offers several potential advantages over conventional architecture; calculations are carried out in parallel yielding speed advantages, and programming can be done by training using examples, rather than defining explicit instructions. Almost all neural-network applications in robot control involve learning the robot system dynamics and incorporating it somehow into the robot controller [7, 8]. The approaches differ in the methods of incorporating the neural network into the controller and of training and adaptation.

One approach is to directly substitute the neural network for the conventional controller in the feedback loop. The neural network is trained off-line, using the

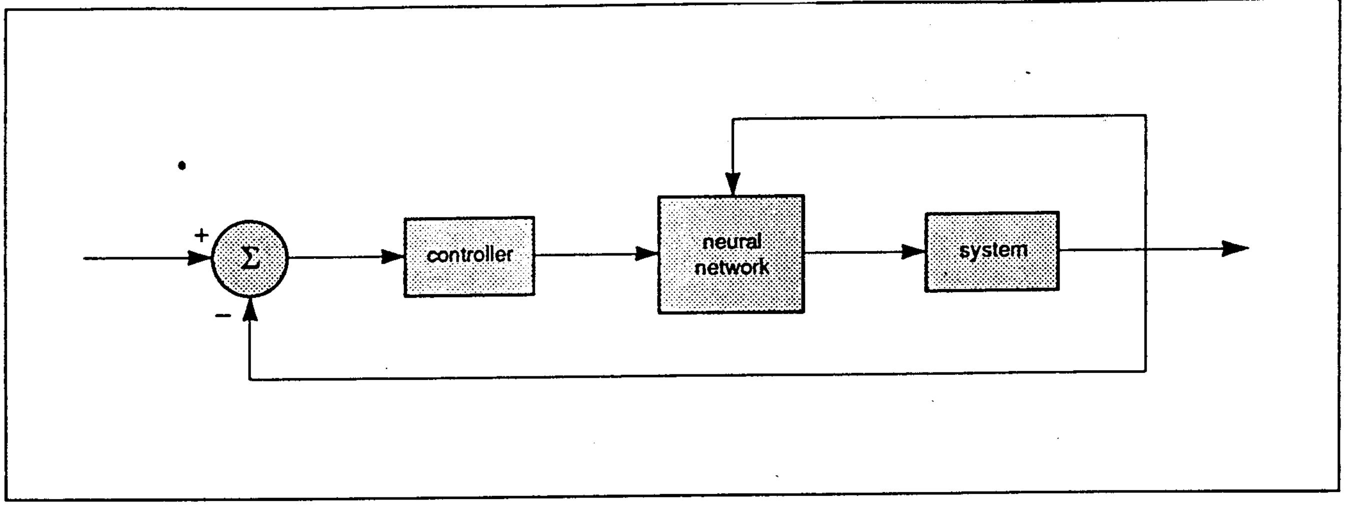

Fig. 2 Neural network as a linearising adaptive controller

input and output of the conventional controller, so that it will give the same output as the conventional controller for the same input. The neural-network controller offers significant speed advantage, given its parallel nature, and could be implemented in real-time even though the implementation of the conventional controller is not possible in real-time if the conventional controller is very complicated.

Another approach is to train the neural network to learn the system dynamics and employ it as a feedforward controller, as shown in Fig. 1. The input to the neural network are the current state of the robot and the required small change from the present state specified by the desired trajectory. The output of the network provides the correction to the force output of the controller. The neural network is continually trained over the space of small changes as the ordinary feedback controller is functioning. Gradually, the neural network takes over the control of the robot as it makes more nearly exact choices and the feedback control functions less. This kind of adaptive control was suggested in Reference 9.

In our approach, we employ the neural network as a replacement for the non-linear robot dynamics evaluation inside the model-based control algorithm. The neural network is embedded in the forward path to 'learn' and then to cancel out the non-linear dynamics of the robot so that the performance of the system appears linear to the controller, as shown in Fig. 2. At each sampling instant, the neural network relearns the robot dynamic model and updates itself. Before we present the details of our neuro-model-based adaptive controller, we first review model-based robot control.

3 Review of model-based control

The dynamic model describing the motion of an N-joint robot is a set of N highly non-linear and coupled differential equations, which relate the actuating joint forces/torques ($\mathbf{f} \in \Re^N$) with the joint positions, velocities, and accelerations ($\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$, $\ddot{\mathbf{q}}(t) \in \Re^N$) respectively [10]:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{f}(t) \tag{1}$$

where $\mathbf{D}(\mathbf{q}) \in \Re^{N \times N}$ is the *positive-definite* inertial matrix, and $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \in \Re^N$ is the coupling vector that incorporates the centrifugal and Coriolis, gravitational and frictional

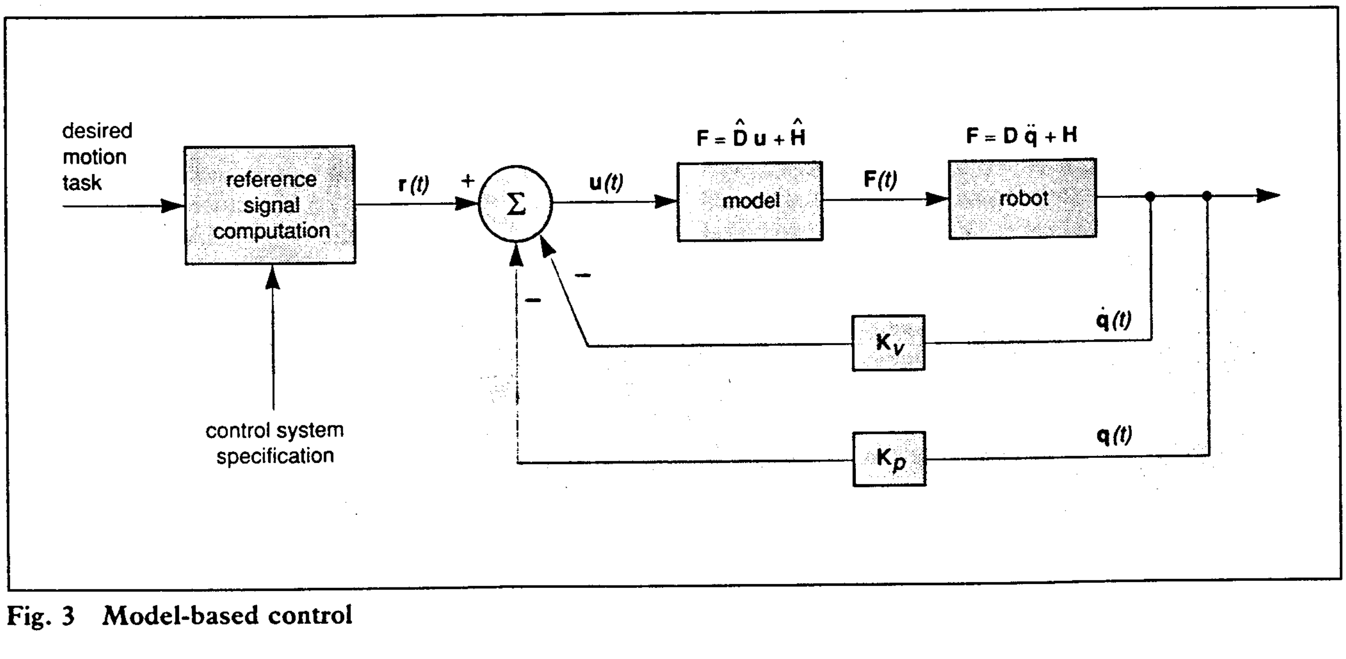

Fig. 3 Model-based control

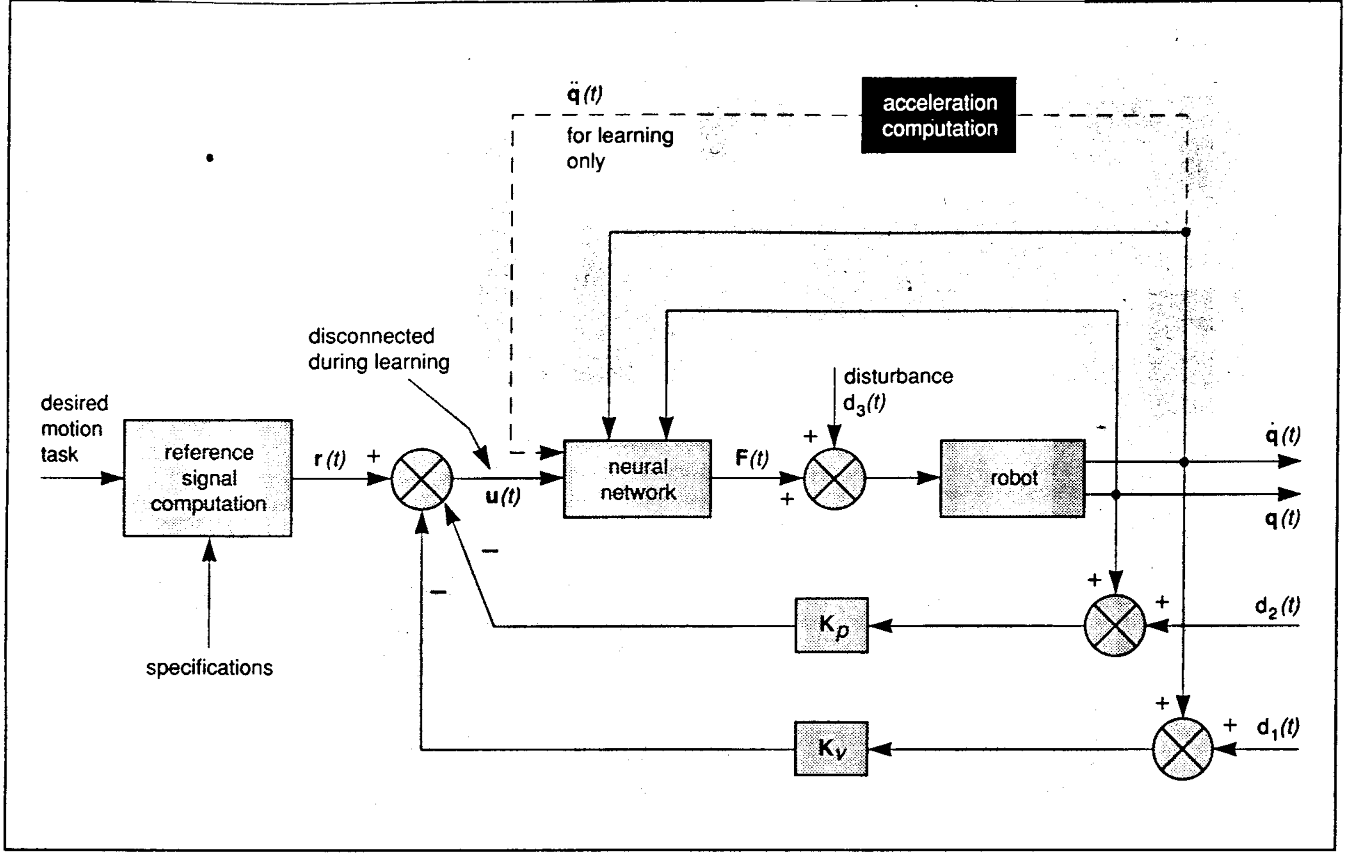

Fig. 4 Neuro-model-based adaptive controller

force/torque vectors. The positive-definiteness of the inertial matrix guarantees that the robot system is completely *state-controllable* [11]; whereas the 2*N*-independent state variables, consisting of joint positions and velocities, render the system completely *state-observable* since these states are measurable.

Model-based robot control involves incorporating the robot dynamic model into the robot controller to transform the highly non-linear robot dynamics into equivalent linear systems. *Linear* control theory can then be applied to synthesise controllers to specify the closed-loop response. Fig. 3 is a schematic diagram of model-based control with proportional-derivative (PD) feedback. The robot is actuated with the following joint force/torque signal:

$$\mathbf{f}(t) = \hat{\mathbf{D}}(\mathbf{q})\mathbf{u}(t) + \hat{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2}$$

where $\mathbf{u}(t)$ is an input signal in the form of acceleration. In eqn. 2, the $\hat{}$ signifies the estimated inertial matrix $\mathbf{D}(\mathbf{q})$ and non-linear coupling vector $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$ implemented in the controller. These estimates (of the robot dynamics) are incorporated in the controller and, because of inevitable modelling and parameter errors, may not be exactly equal to the actual $\mathbf{D}(\mathbf{q})$ and $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$.

If the robot dynamics are modelled *perfectly*, i.e. if $\mathbf{D}(\mathbf{q}) \leftarrow \hat{\mathbf{D}}(\mathbf{q})$ and $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \leftarrow \hat{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}})$ in eqn. 2, then equating eqn. 2 with eqn. 1 results in the following linear system:

$$\ddot{\mathbf{q}} = \mathbf{u}(t) \tag{3}$$

The resulting linear system in eqn. 3 allows the specification of the necessary feedback gains, as well as the required reference signal, to perform a robotic motion task. There are many possible choices for the reference signal, depending on the desired closed-loop system response. For perfect tracking in a PD implementation, the reference signal is specified according to the *computed-torque* algorithm [12]:

$$\mathbf{r}(t) = \ddot{\mathbf{q}}_d(t) + \mathbf{K}_v\dot{\mathbf{q}}_d(t) + \mathbf{K}_p\mathbf{q}_d(t). \tag{4}$$

where *d* indicates the desired trajectory. The acceleration input signal becomes

$$\begin{aligned}\mathbf{u}(t) &= \mathbf{r}(t) - \mathbf{K}_v\dot{\mathbf{q}}(t) - \mathbf{K}_p\mathbf{q}(t)\\ &= \ddot{\mathbf{q}}_d(t) + \mathbf{K}_v(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}).\end{aligned} \tag{5}$$

Substituting eqn. 5 into eqn 3 yields

$$\ddot{\mathbf{q}} + \mathbf{K}_v\dot{\mathbf{q}} + \mathbf{K}_p\mathbf{q} = \ddot{\mathbf{q}}_d + \mathbf{K}_v\dot{\mathbf{q}}_d + \mathbf{K}_p\mathbf{q}_d \tag{6}$$

or

$$\ddot{\mathbf{e}} + \mathbf{K}_v\dot{\mathbf{e}} + \mathbf{K}_p\mathbf{e} = 0 \tag{7}$$

where $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$ is the *tracking error* vector. Taking the Laplace transform of eqn. 6 results in a unity transfer function between the output trajectory $\mathbf{q}(t)$ and the input trajectory $\mathbf{q}_d(t)$, and this demonstrates perfect tracking. Eqn. 7 is the so-called *error equation*; what it means is that, using the reference signal (eqn. 4), we can decide how the tracking error goes to zero *asymptotically* by specifying appropriate values of $\mathbf{K}_v$ and $\mathbf{K}_p$.

Diagonal position and velocity gain matrices are typically specified to decouple the axes of the closed-loop system in eqn. 7:

$$\mathbf{K}_p = k_p\mathbf{I} \quad \text{and} \quad \mathbf{K}_v = \mathbf{K}_v = k_v\mathbf{I} \tag{8}$$

where k_p and k_v are scalar feedback gains and $\mathbf{I} \in R^{N \times N}$ is the identity matrix. (In practice, the designer can select different position and velocity feedback gains for each joint.)

The feedback gains k_p and k_v are specified to guarantee the stability of eqn. 7. We implement the *critically damped* design

$$k_v^2 = 4k_p \Leftrightarrow k_v = 2\lambda > 0 \quad \text{and} \quad k_p = \lambda^2 \tag{9}$$

which provides the fastest possible response without overshoot. In eqn. 9, λ is the control bandwidth of the closed-loop system.

4 Neuro-model-based controller

Model-based robot control has many practical problems. The computational complexity involved in evaluating the robot dynamic model has led to significant simplifications in the robot dynamic model so that it can be implemented in the control servo loop. In addition, a mathematical model that *completely* and *accurately* characterises the robot dynamic behavior is, in general, not realisable. Not all physical phenomena can be modelled accurately; examples include friction, backlash and hysteresis effects. Furthermore, the robot dynamic behaviour typically changes as 'wear and tear' sets in.

These limitations of model-based robot control have stimulated the use of neural networks. The inherently parallel architecture of neural networks allows for real-time implementation of complex robot control algorithms. The 'trainability' of neural networks allows on-line learning of the (changing) robot dynamic behaviour. Another interesting characteristic of neural networks is '*generalisability*', i.e. the capability to learn new ideas (e.g. mappings, relationships etc.) from input data the networks have not previously seen. This characteristic, when properly implemented, makes neural-network-based robot controllers robust towards disturbances such as load disturbances and measurement noise.

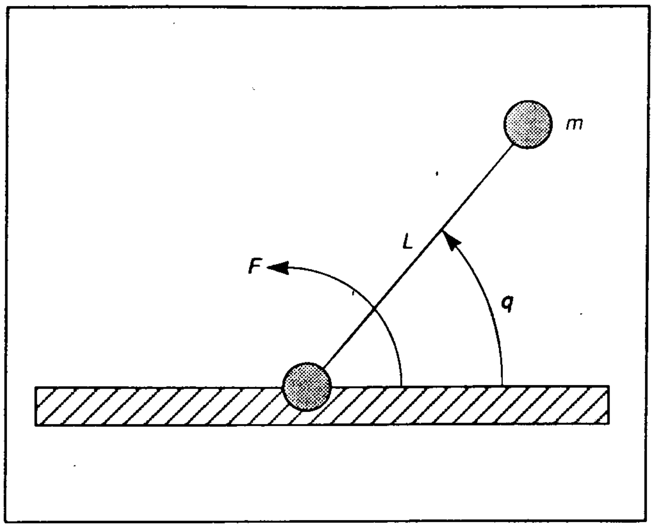

Fig. 5 One-dimensional robot model

There are many kinds of neural networks. For our application, the most appropriate is the *multilayer feed-forward* neural network, which learns using the *back-propagation* algorithm [6]. Such networks are designed to learn non-linear function mappings, such as the non-linear function describing the mapping from joint positions, velocities and accelerations to (actuating) joint forces/torques (eqn. 1). In our framework, we employ the multilayer feedforward network to evaluate, in real time, the non-linear robot dynamics model. In addition, at each sampling instant, the neural network learns and updates its knowledge of the robot dynamic model. For robustness, the proportional-derivative (PD) block of conventional model-based control is retained as the closed-loop controller and provides the acceleration command signal to the neural-network dynamics evaluator.

Fig. 4 shows the schematic diagram of the neuro-model-based control system. The neural network's task

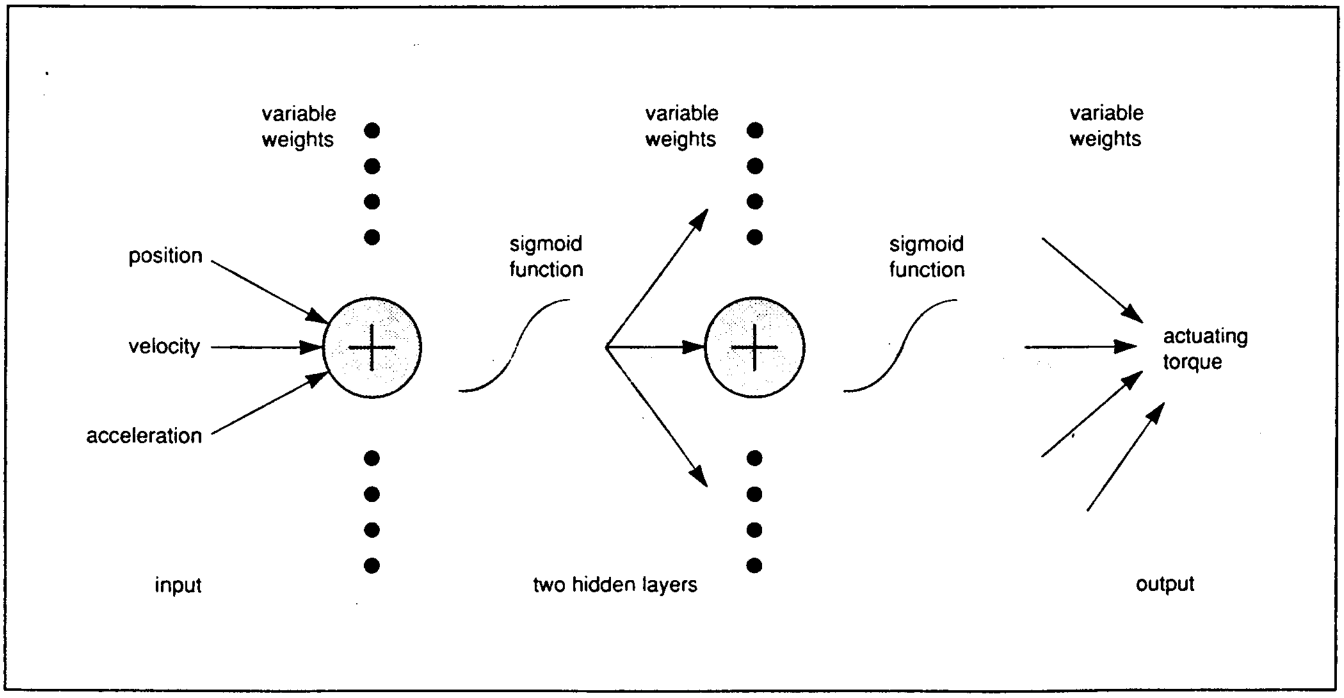

Fig. 6 Neural network for dynamics evaluation

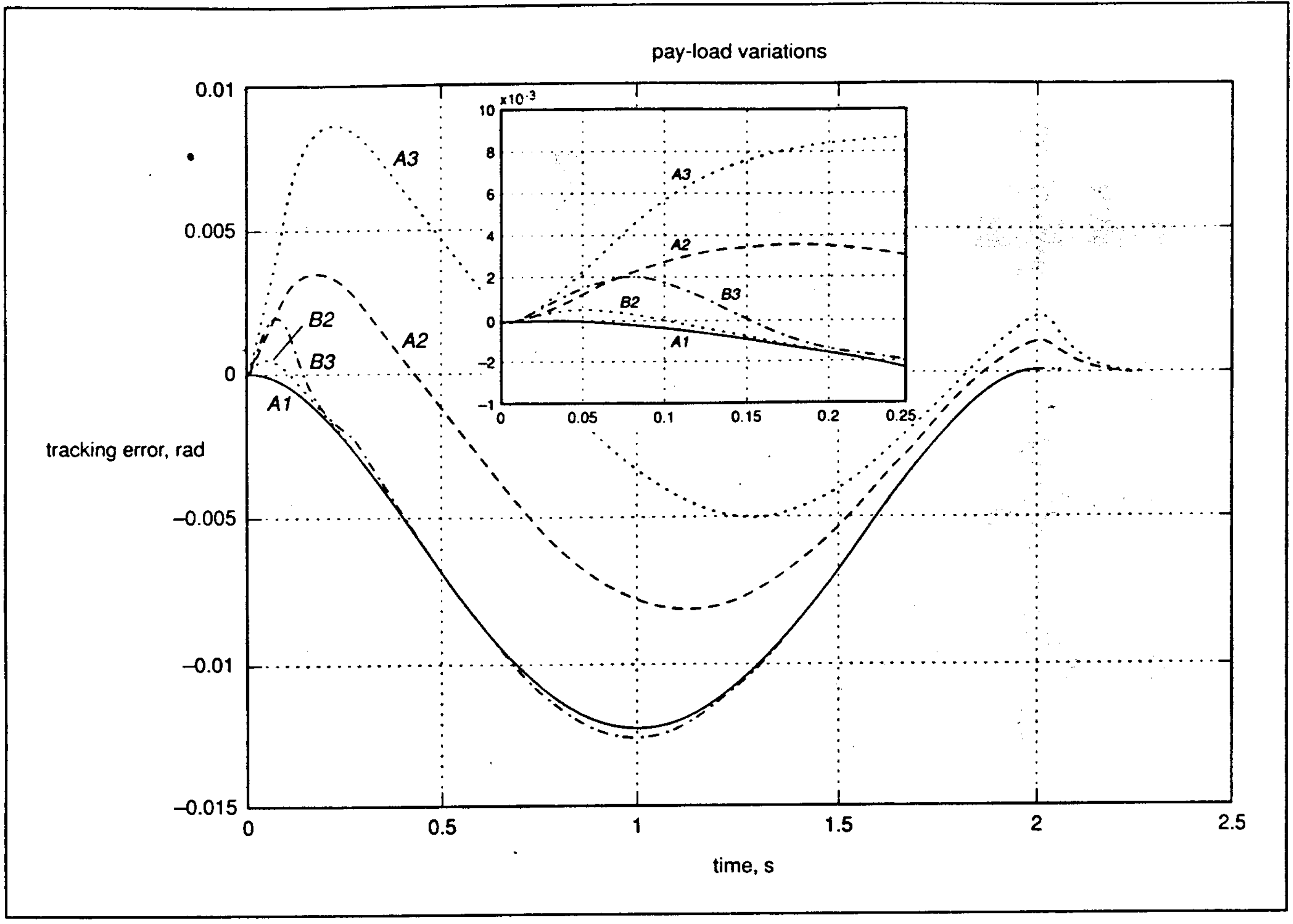

Fig. 7 Tracking error due to pay-load variation

Table 1 Errors due to pay-load mass variations

curve	nominal mass, kg	true mass, kg	final error (at $t_f = 2$ s)
A1 (solid)	0.10	0.10	0.000075
A2 (dashed)	0.10	0.12	0.001042
A3 (dotted)	0.10	0.14	0.001997
B1 (solid)	0.10	0.10	0.000086
B2 (dotted)	0.10	0.12	0.000117
B3 (dashdot)	0.10	0.14	0.000168

is to continuously learn the robot dyanamic model. It has three sets of input consisting of the N joint positions, velocities and accelerations. The output is a vector of the N joint forces/torques. Training data for this network can easily be generated using the nominal dynamic parameters in eqn. 1.

We propose the incorporation of the network into the robot controller to emulate the on-line dynamics evaluation. Fig. 4 shows the schematic diagram of the neuro-model-based controller. The solid lines show the signal flow wherein the neural network is executing the control action, and the shaded lines represent the information flow during learning. The first stage involves control execution, and the second stage is the learning phase. During learning, the input to the neural network consists of the current state of the robot, as well as the signal from the acceleration computation block, instead of the control input signal $\mathbf{u}(t)$.

Depending on the control system specification, the reference signal $\mathbf{r}(t)$ is computed from the desired robot motion. A PD control imlementation dictates the computed torque algorithm (eqn. 4) for the specification of the reference signal. An input signal in the form of acceleration $\mathbf{u}(t)$, as well as the current state of the robot, is fed into the neural network to generate a force/torque command $\mathbf{f}(t)$ to the robot. This actuation brings the robot to a new state, from which the acceleration *corresponding* to the actuating force is computed (using a finite difference approximation). The acceleration, together with the new robot state, provides the learning input and the corresponding actuating force is the learning output. This training information is used by the neural network to adapt to the robot behaviour.

5 Simulation studies

The plant we chose to control is a one-dimensional robot wherein gravity provides the non-linearity. Fig. 5 is a schematic diagram of the robot system. The pay-load is considered as a point mass m concentrated at the end of the link, and the link of length L is assumed to be massless. The input to the robot is the driving torque F, and the state of the system is given by the joint position q and velocity $\dot{q}$. The robot dynamic model is then

$$F = mL^2\ddot{q} + mgL\cos q \qquad (10)$$

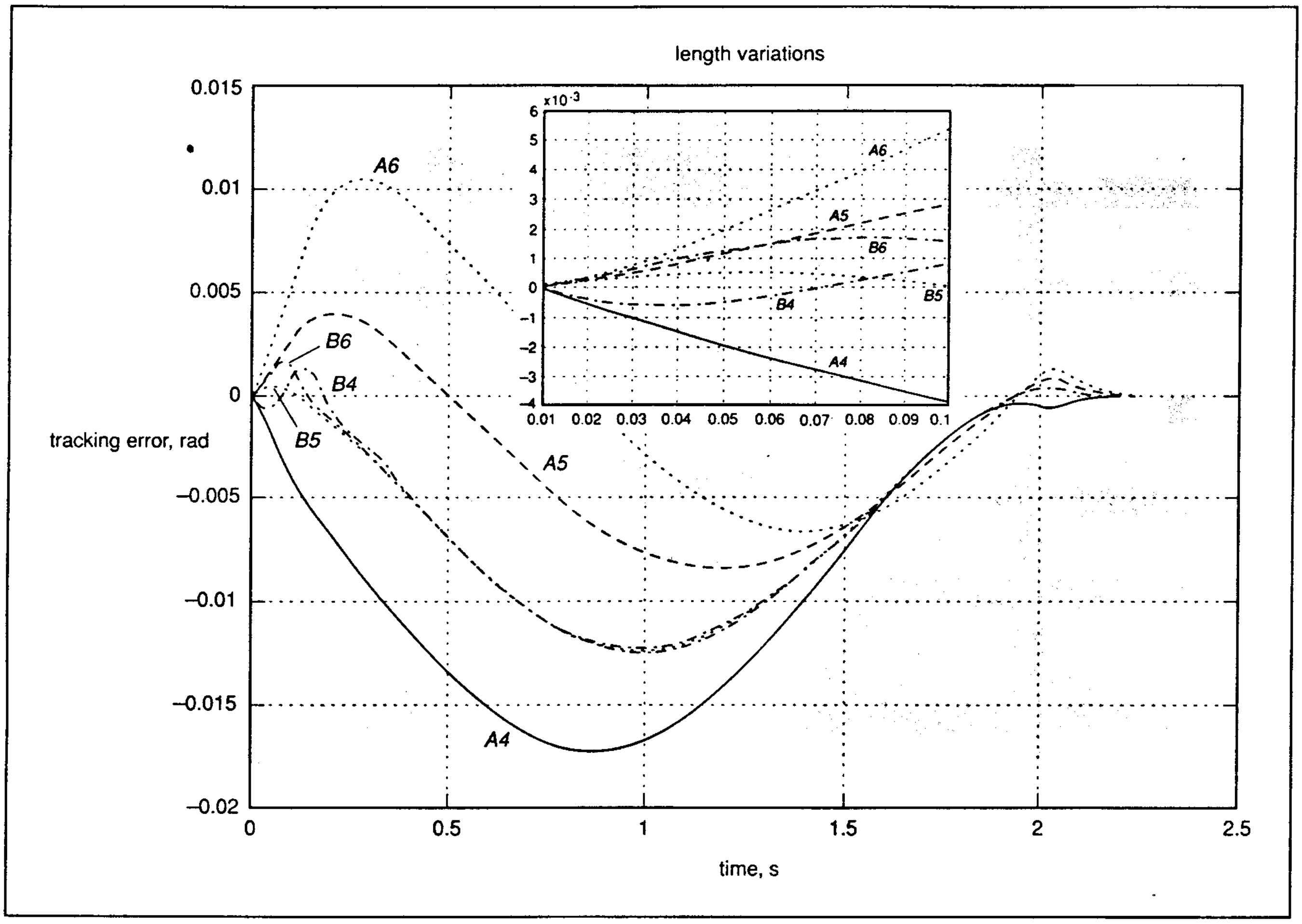

Fig. 8 Tracking error due to length variation

where g is the acceleration due to gravity. For purposes of numerical simulations, the parameters of the robot were chosen to be $m=0.10$ kg and $L=1$ m.

5.1 Pre-training the neural network

We use the back-propagation network with two hidden layers (Fig. 6) to learn the robot dynamic model. Each neuron in the hidden layer has a sigmoid decision function of 'slope 1'. The input is the joint position and acceleration, and the output is the joint torque. There are two inputs consisting of two neurons, for which the input is the point position q and acceleration $\ddot{q}$. The output layer consists of one neuron, for which the output is the required torque.

Before implementing the neuro-model-based robot controller, the network is pre-trained using the nominal dynamic parameters ($m=0.10$ kg and $L=1$ m) of the robot. The training input consists of uniformly distributed random values of joint position and acceleration in the following ranges:

joint position	$0 \leq q \leq \pi$ rad
joint acceleration	$-3 \leq \ddot{q} \leq 3$ rad/s^2

The training output consists of the torques computed using eqn. 10 from the training input.

For expository convenience, we refer to a network as $N_1 - N_2$ if there are N_1 and N_2 neurons in the first (from the input) and second hidden layers, respectively. We tried networks of size 5-5, 10-5, 15-5, 20-10, 20-15 and 30-20 with training in the order of one million times. The weights in the neural network were updated using the delta rule, with learning and momentum coefficients of 0.4 and 0.2, respectively. The 15-5 network provides the most accurate emulation of robot dynamics with a maximum torque deviation of 0.0015 Nm for a torque range of ± 1.28 Nm. Increasing the number of neurons does not result in a more accurate dynamics emulation. For the controller simulation, we therefore use the pre-trained 15-5 network.

The neural network was also trained with data corrupted by random noise to simulate actual experimental data. The training data used were the same as before except that random noise was added to the desired output torque. The output presented to the net during training is thus corrupted by the random noise.

Table 2 Errors due to length variations

curve	nominal length, m	true length, m	final error (at $t_f=2$ s)
A4 (solid)	1.0	0.80	−0.000650
A5 (dashed)	1.0	1.20	0.000615
A6 (dotted)	1.0	0.40	0.000923
B4 (dashdot)	1.0	0.80	0.000064
B5 (dotted)	1.0	1.20	0.000152
B6 (dashdot)	1.0	1.40	0.000235

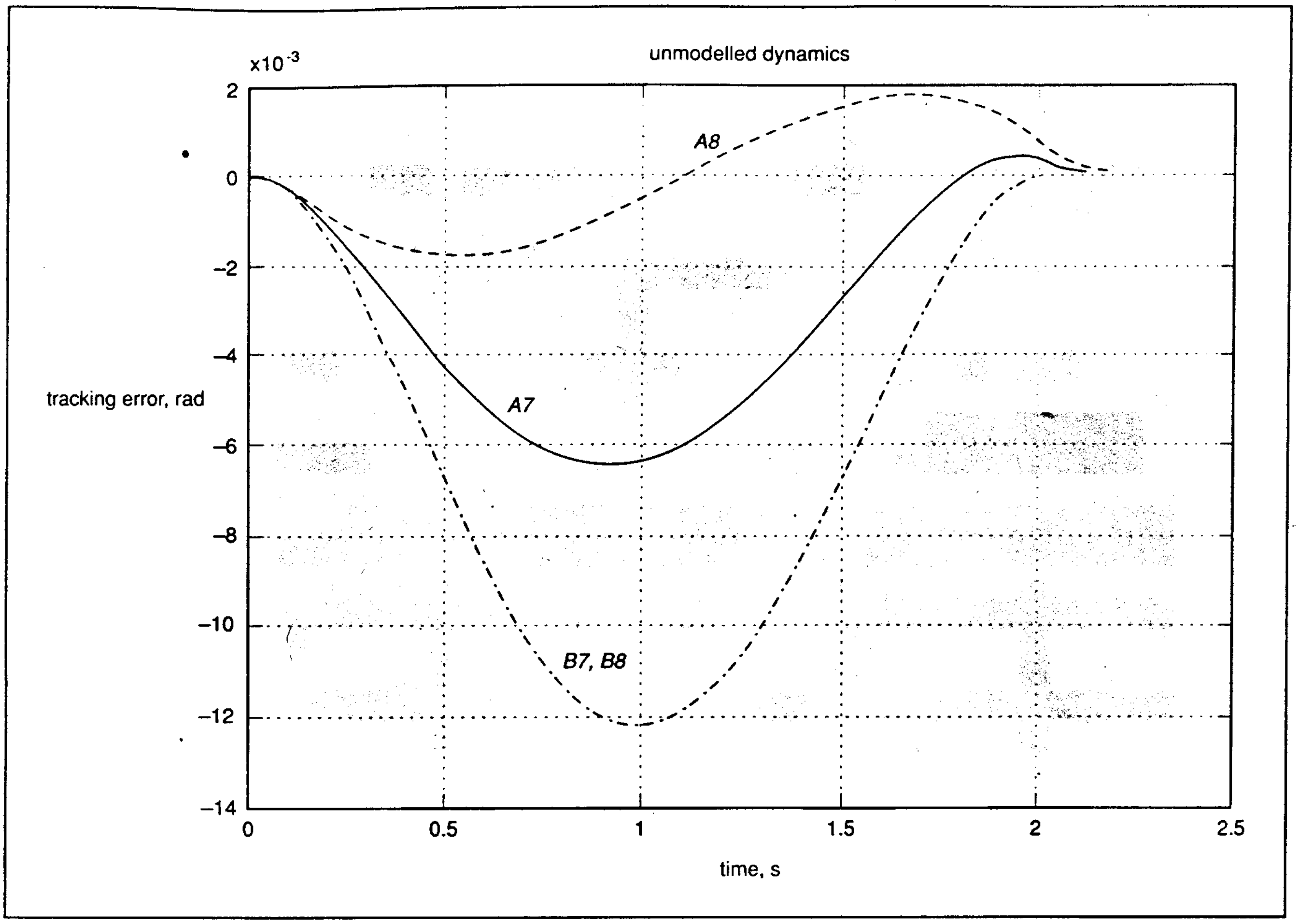

Fig. 9 Tracking error due to unmodelled dynamics

Table 3 Errors due to unmodelled dynamics

curve	damping coefficient	final error (at $t_f = 2$ s)
A7 (solid)	0.2	0.000348
A8 (dashed)	0.4	0.000680
B7 (dashdot)	0.2	0.000023
B8 (dashdot)	0.4	−0.000005

5.2 Desired motion task

The robot is commanded to move from $q=0$ to $q=q_f=1.3$ radians in exactly $t_f=2$ s. The initial and final velocities and accelerations are all zero. To satisfy the boundary conditions, a quintic polynomial is used for the trajectory. This trajectory [13] is

$$q(t) = 10\frac{q_f}{t_f^3}t^3 - 15\frac{q_f}{t_f^4}t^4 + 6\frac{q_f}{t_f^5}t^5 \tag{11}$$

The reference signal to the controller is then computed according to eqn. 4.

It should be emphasised that the task is for the robot to arrive at the final joint position $q_f = 1.3$ rad in exactly $t_f = 2$ s. The quintic polynomial trajectory is just a means of specifying the reference signal to ensure a smooth torque profile and compliance with boundary conditions. Therefore, for this task, the important performance indicator is the final error at $t_f = 2$ s.

5.3 Neuro-model-based control

The pre-trained 15-5 network is employed in the robot controller, as shown in Fig. 4. The feedback gains are specified as $k_v = 40s^{-1}$ and $k_p = 400$ for a critically damped design with a bandwidth of $20s^{-1}$. The sampling interval was chosen as 10 ms. The reference signal and the weights of the neural network are updated once every 10 ms. The robot motion is simulated by numerically solving eqn. 10, using a fourth-order adaptive Runge-Kutta algorithm with a maximum error specification of 10^{-6} rad [14].

We simulate modelling errors in the form of parameter errors (i.e. mass m and length L variations) and unmodelled dynamics in the form of unmodelled velocity damping effects. For purposes of benchmark comparisons, we also provide results for the 'standard' model-based controller implementation.

The modelling errors were simulated by using the nominal dynamic parameters ($m = 0.10$ kg, $L = 1.00$ m, no damping) in the on-line dynamics evaluation (Fig. 3), and the robot plant is simulated using the *true* dynamic model. To simulate the velocity damping effect, the robot plant model used is

$$F = mL^2\ddot{q} + mgL\cos q + \delta\dot{q} \tag{12}$$

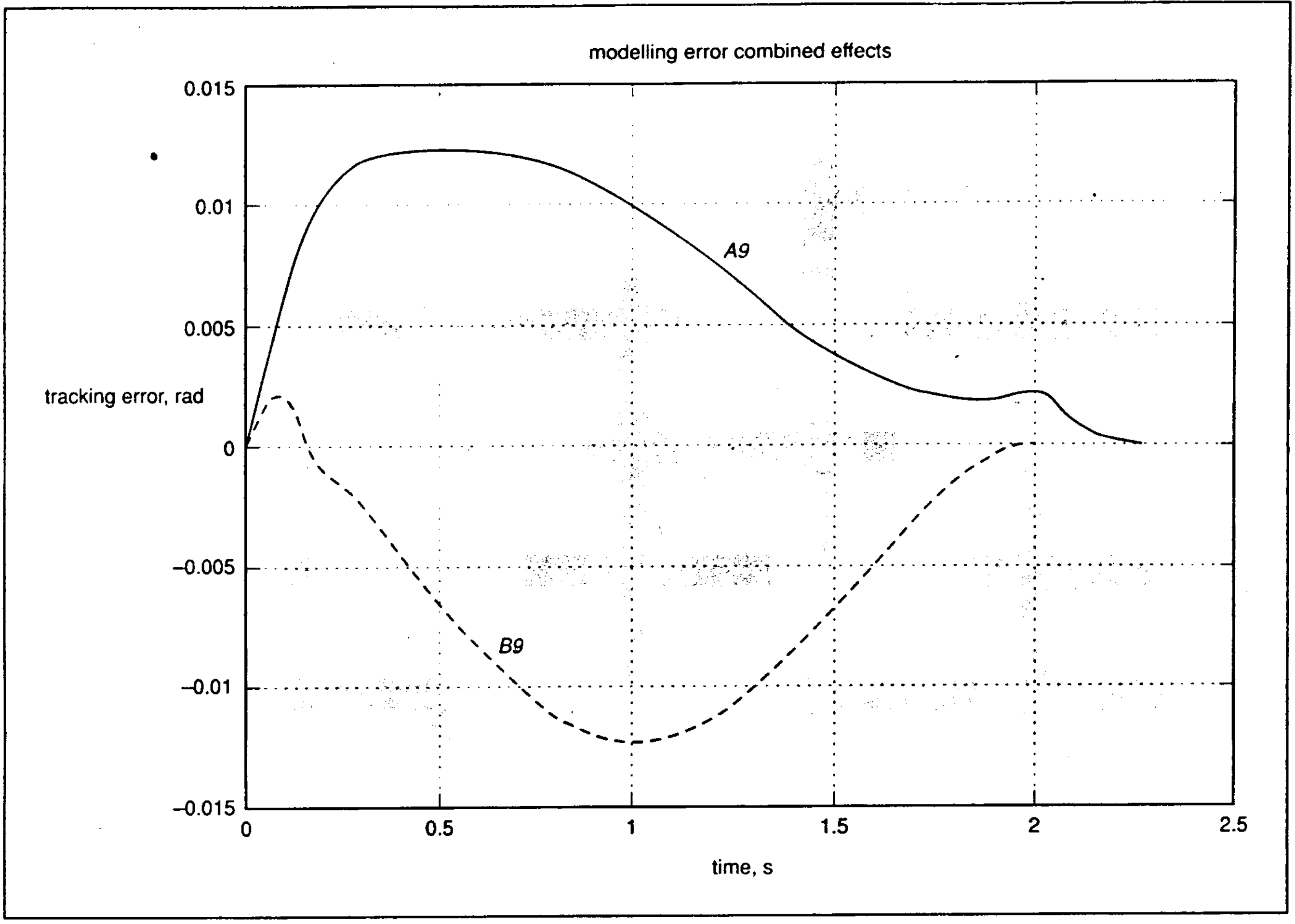

Fig. 10 Tracking error due to modelling errors

To see the effects of disturbances, disturbances were injected at the input (input noise d_3) to the robot and at the output (measurement noises d_1, d_2) of the system (Fig. 4). Gaussian white noise was used as it occurs frequently in practice. The disturbances are assumed to be unmeasurable, and no other knowledge (of the distrubances) is assumed other than that it is Gaussian.

5.3.1. Pay-load mass variations: we simulate pay-load mass variations of up to 40%, with $L = 1.00$ m and no damping. The plots of the trajectory tracking error e versus time are shown in Fig. 7. Curves *Ai* represent the performance of standard model-based control, and curves *Bi* are for the neuro-model-based control. Table 1 indicates the meaning of the curves and numeric results. Curve *A1* represents perfect knowledge of the robot dynamic model, whereas curve *B1* is for the neuro-model-based controller pre-trained on a non-time-varying plant; these two curves are indistinguishable (Fig. 7). It can be observed that the actual joint trajectory leads the desired joint trajectory (negative tracking error) for exactly half of the duration of the move (1 s). During this time, the joint acceleration is continuously increasing, in anticipation of the inevitable lag when tracking a step input during each sampling interval. For the remaining half, the joint acceleration starts to decrease until it is zero at $t_f = 2$ s, and the actual trajectory approaches the desired trajectory at the end of the move. During this time, the accumulated 'overshoot' in the first half of the move was compensated for by the tracking lag. Of course, this behaviour is only possible with perfect knowledge of robot dynamics.

The tracking error curves fail to follow the 'standard' inverted bell shape in the presence of modelling errors. The adaptation capability of the neural network, however, allows fast compensation of modelling errors (of the order of 10–20 steps) to reach the final destination with close to ideal performance. Curves *B2* and *B3* show a reduced error trajectory, as well as a smaller final error at $t_f = 2$ s.

5.3.2 Length variation: the results for length variations of up to 40%, with $m = 0.10$ kg and no damping, are shown in Fig. 8. Here similarly, curves *Ai* represent the performance of standard model-based control, and curves *Bi* are for our neuro-model-based control. Table 2 highlights the numerical results. For all cases, the neuro-model-based controller (curves *Bi*) has the lower error trajectory curves.

5.3.3 Unmodelled velocity damping effects: velocity damping coefficients of 0.2 and 0.4 were simulated with $m = 0.10$ kg and $L = 1.00$ m. The results are shown in Fig. 9 and Table 3. Curves *B7* and *B8* are for the neuro-model-based control, and they follow a similarly shaped curve for the performance of model-based control with exact knowledge of robot dynamics (curve *A1* in Fig. 7). This demonstrates how well the neural network is adapting to the robot plant. It is interesting to note that curves *A7* and *A8* demonstrate smaller errors than *B7*

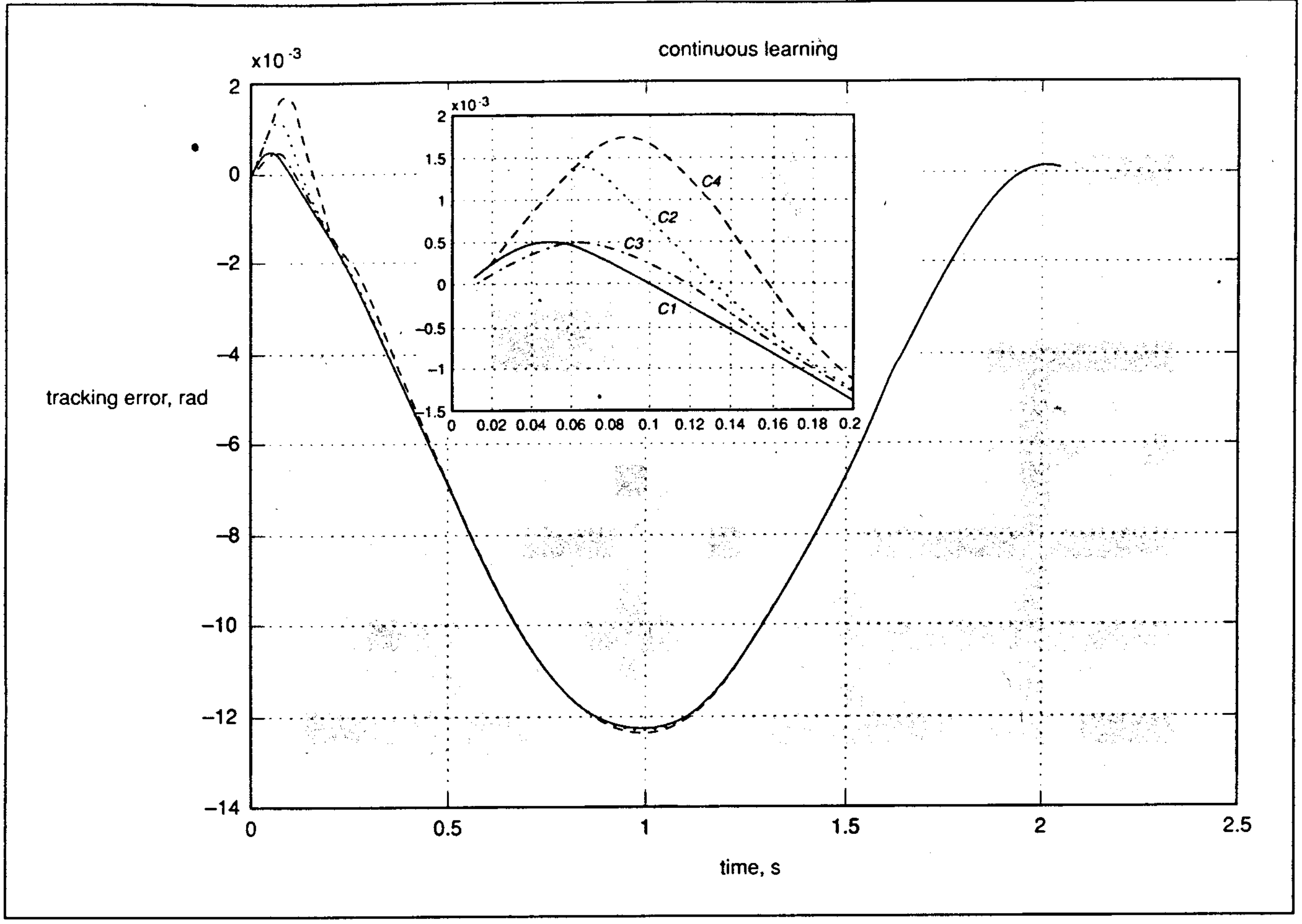

Fig. 11 Tracking error improvement through continuous learning

and *B8*, but larger final errors (at $t_f = 2$ s). In fact, model-based control with modelling errors is better than with exact knowledge of robot dynamics. This behaviour, however, depends on the desired task trajectory.

5.3.4 Combined effects of modelling errors: Fig. 10 shows the results for parameter errors as well as unmodelled velocity damping effects. The true parameters are $m = 0.12$ kg, $L = 1.2$ m and $\delta = 0.4$ Nms. Curve *A9* for model-based control has a final error of 0.002109 rad, and curve *B9* for neuro-model-based control has a final error of 0.000047 rad.

5.3.5 Continuous learning: to further demonstrate the fast adaptation of our neuro-model-based control, we simulate the controller's improved performance over a sequence of tasks. Four tasks (*C1–C4*) are executed in the indicated sequence, and the controller continuously updates its knowledge of the robot plant. Tasks *C1* and *C2* involve a mass error of 20% ($m = 0.12$ kg) and 40% ($m = 0.40$ kg), respectively. Task *C3* is exactly the same as *C2*, and therefore an improved performance is expected for tracking *C3* since it is executed after *C2*. The results are shown in Fig. 11. We note that *C3* has a smaller trajectory tracking error compared to *C2*. Furthermore, the final error (at $t_f = 2$ s) dropped from 0.000141 rad for *C2* to 0.000127 rad for *C3*. Task *C4* involves model parameters of $m = 0.15$ kg, $L = 1.1$ m and $\delta = 0.1$ Nms. For all the curves, the maximum final error at $t_f = 2$ s is 0.000141 rad.

5.3.6 Effects of disturbances: tachometer sensor disturbance d_1 (Fig. 4) was simulated by adding Gaussian noise of unit variance with magnitude scaled by 10%. The response of the system is shown in Fig. 12, where the solid curve represents the desired position trajectory and curve B_{NET1} the actual path trajectory. Although the trajectory is smooth, it can be seen that there is a significant final positional error. The results are also shown in Table 4. A disturbance d_2 is next introduced in the inner feedback loop, as shown in Fig. 4. d_2 has unit variance and magnitude scaled by 80%, and so the response of the system is as shown in Fig. 13. It can be noted that, in this case, there is also a significant final positional error. The reason for having different d_1 and d_2 is because since $k_p = 400$ and $k_v = 40$, in order to observe similar output effects in the presence of d_1 and d_2, magnitudes for $k_p d_1$ should be approximately similar to that for $k_v d_2$.

In this case, it would seem that the neural-network controller is unable to handle a system with noisy signals.

Table 4 Final position errors

curve	10% d_1	80% d_2	10% d_1 & 20% d_3
B_{NET1}	0.36366	0.16585	0.77429
B_{NET2}	0.00061	0.00858	0.06495

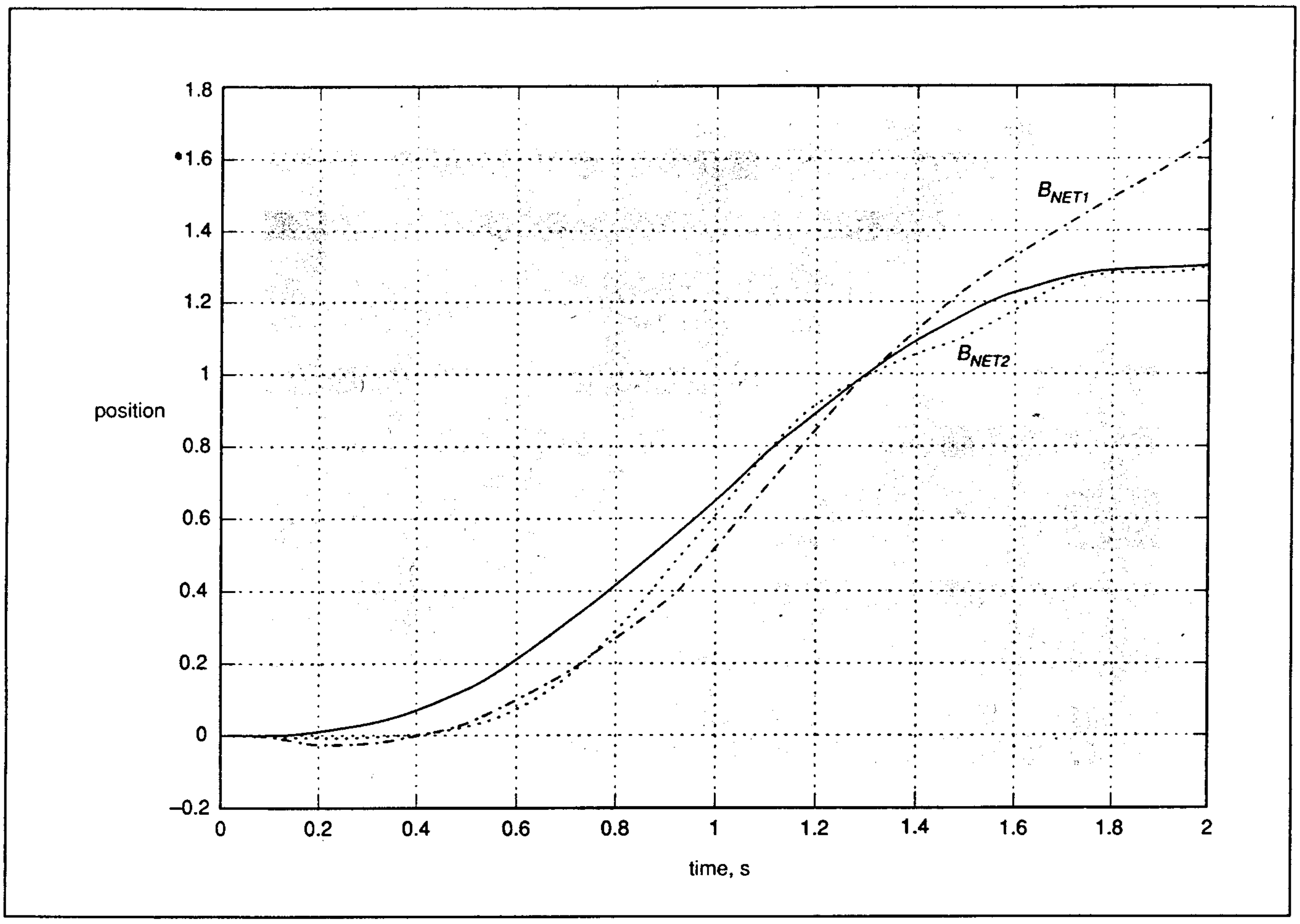

Fig. 12 Tracking error in the presence of outer feedback loop disturbance: comparison of *NET*1 and *NET*2

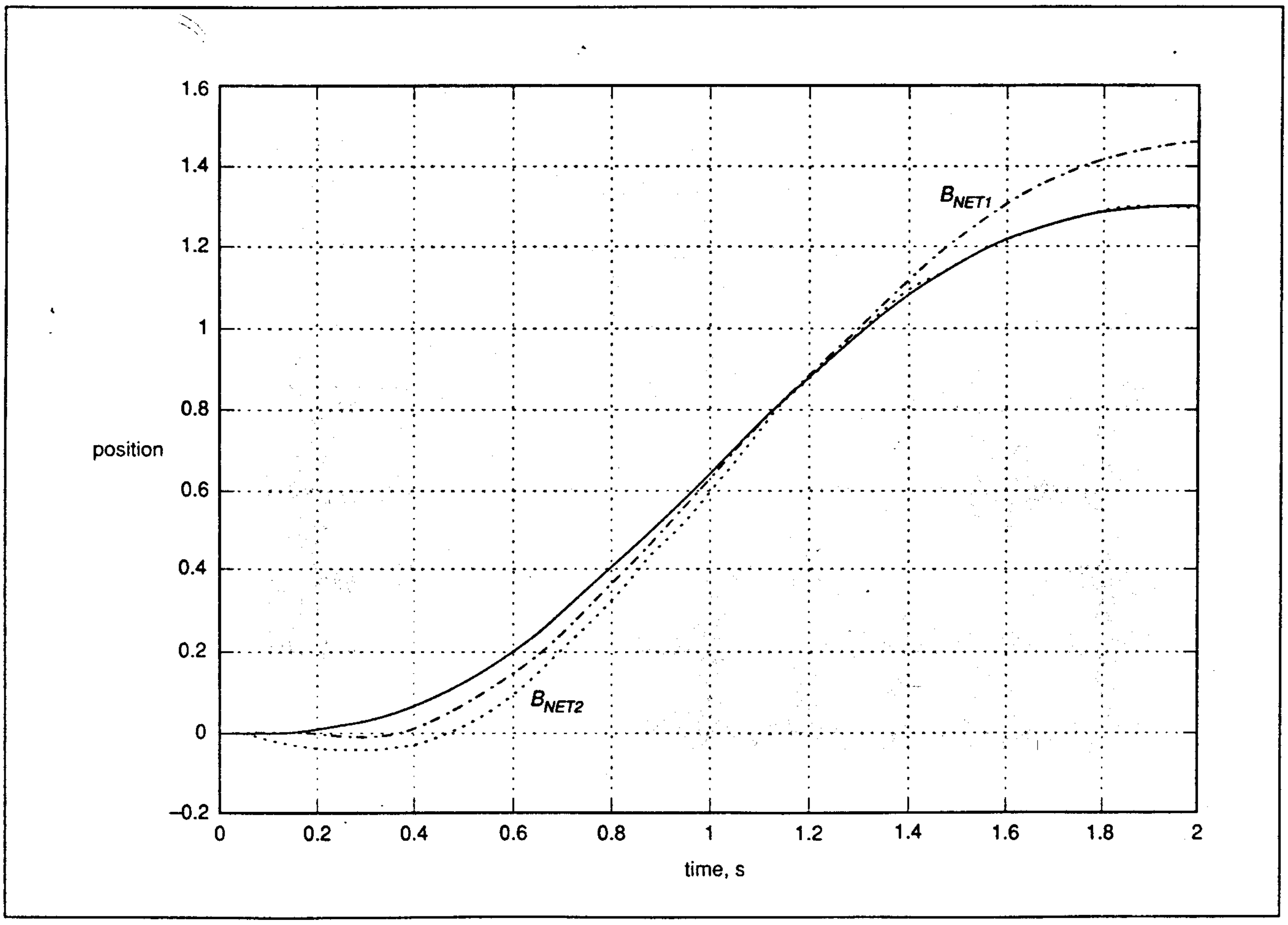

Fig. 13 Tracking error in the presence of inner feedback loop disturbance: comparison of *NET*1 and *NET*2

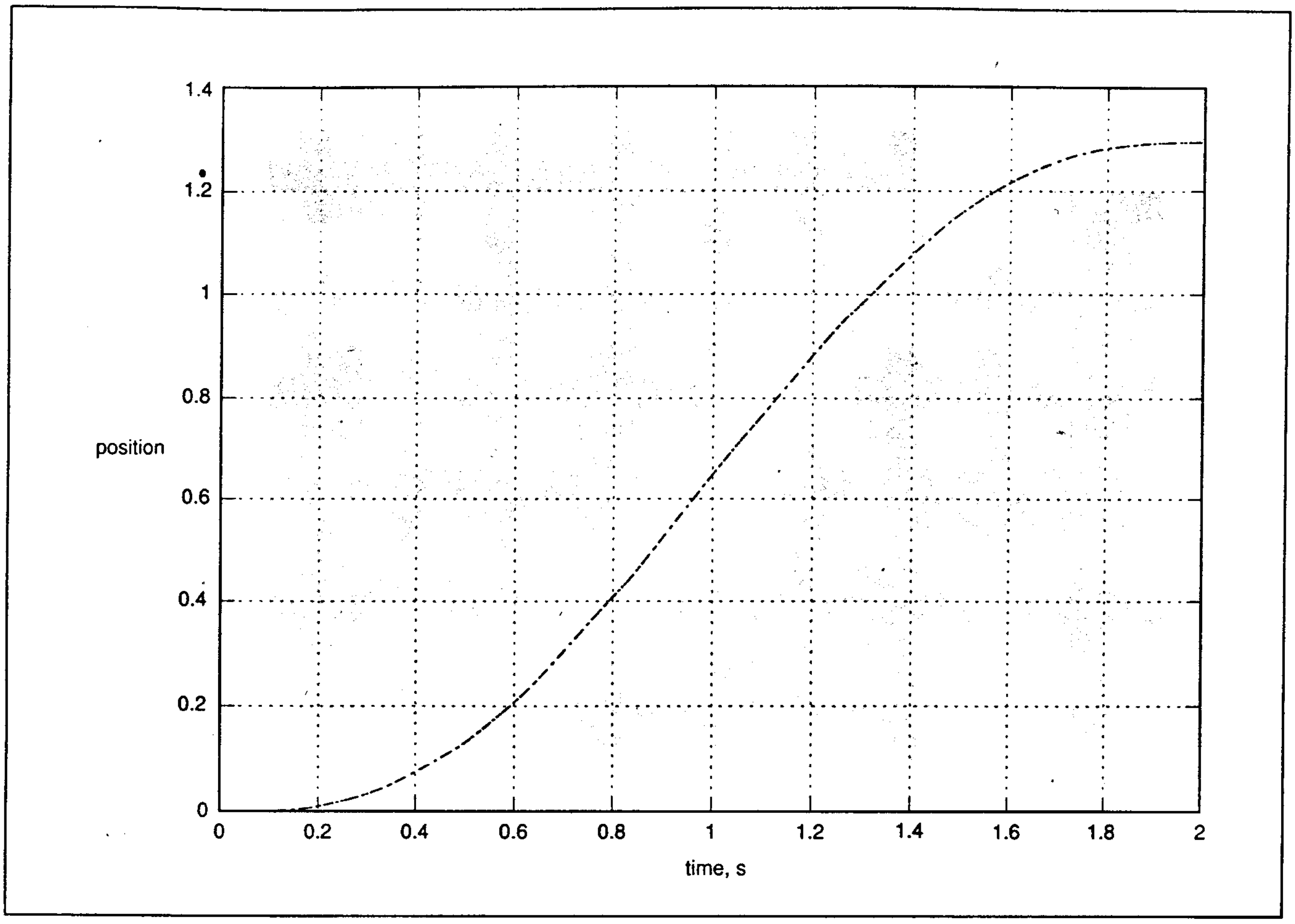

Fig. 14 Tracking path in the absence of noise: comparison of *NET*1 and *NET*2

We next try using the network trained with noise (i.e. *NET*2). The performance of *NET*2, compared to *NET*1, in the absence of noise was first investigated. The results are shown in Fig. 14. It can be seen that the output of the system with either controller can hardly be distinguished from the desired trajectory. This shows that the system with either controller can follow the desired trajectory when there is no noise interference in the system.

Next, the disturbances d_1 and d_2 with the same values as before were introduced into the system. The tracking path obtained with *NET*2 is shown by the dotted lines in Figs. 12 and 13. It can be seen that with *NET*2 the final positional error has been greatly reduced while a smooth trajectory is maintained.

Torque disturbance d_3 (Fig. 6) is simulated by adding noise of unit variance and magnitude scaled by 20% to the input of the robot. Fig. 15 shows the results when disturbance d_1 and disturbance d_3 were present at the same time. Here, it can be seen that the response trajectory (curve *B*), obtained with the neuro-model-based controller, although having a path-tracking error, nevertheless had very little final positional error and had a smooth trajectory. The model-based controller (curve *A*), however, did not perform as well in the presence of the introduced noise. The response trajectory obtained in this case had a significant final positional error, in addition to a rather jerky trajectory path.

6 Conclusions

In this paper, we presented a new approach towards adaptive robot control using the neural-network paradigm. We have demonstrated that a back-propagation network can be trained to 'learn' the highly non-linear robot dynamic model. We implement the neural network into the model-based robot controller to replace the on-line dynamics evaluation. The neural network, being inherently parallel in nature, has the potential to evaluate robot dynamics in real time. Furthermore, the learning capability of the neural network makes it ideal for adaptive robot control. In addition, we have shown that neural networks are tolerant of noise and disturbances.

The results of our simulation show that once a neural network has been trained to learn the robot dynamic model, it can easily adapt or be retrained to learn any slight perturbations of the model parameters. We have demonstrated the robustness of the neuro-model-based controller towards up to 40% modelling errors. It was also found that a neuro-model-based controller trained with noisy data performs better than a conventional model-based controller in the presence of noise. It is interesting to note that in the absence of noise, the former also matches the performance of the latter. Further investigation into this phenomenon is necessary in order to exploit the full benefits of using neural-network controllers.

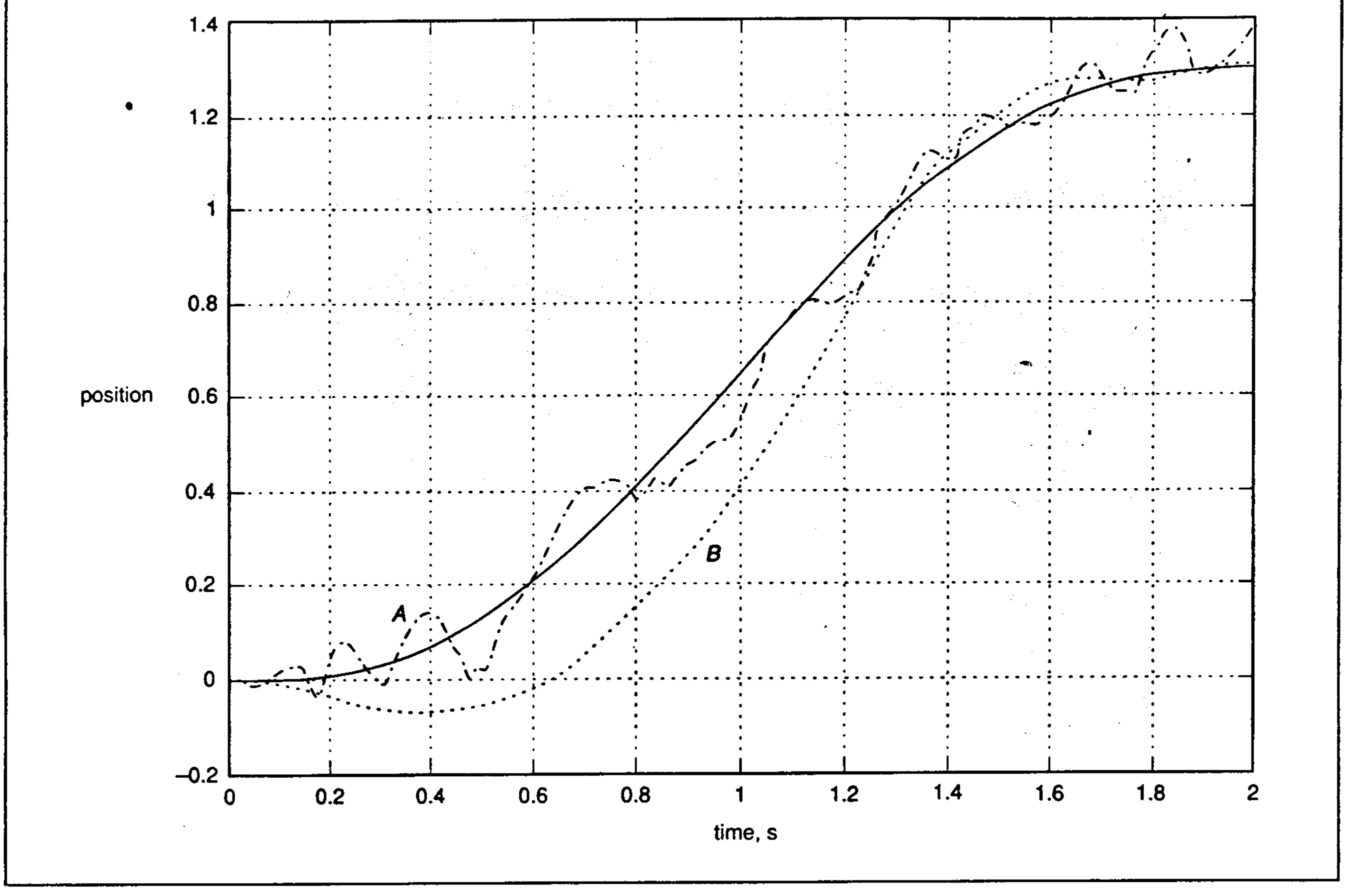

Fig. 15 Tracking path in the presence of output feedback loop disturbance and plant input disturbance: comparison of *NET2* and model-based controller

This study clearly indicates the promise and potential of neural networks in the practical controller implementations for real systems.

7 References

[1] FU, K. S., GONZALEZ, R. C., and LEE, C. S. G.: 'Robotics: control, sensing, vision, and intelligence' (McGraw-Hill, New York, 1987)

[2] TOURASSIS, V. D.: 'Principles and design of model-based controllers', *Int. J. Control*, 1988, **47**, (5), pp. 1267–1275

[3] ASTROM, K. J.: 'Theory and applications of adaptive control: a survey', *Automatica*, 1983, **19**, (5), pp. 471–486

[4] TOURASSIS, V D.: 'Computer-control of robotic manipulators using predictors'. Proc. 1987 IEEE Symp. on Intelligent Control, Philadelphia, Pennsylvania, 18–20 January 1987, pp. 204–209

[5] TOURASSIS, V. D., and NEUMAN, C. P.: 'Robust nonlinear feedback control for robot manipulators', *IEE Proc.* D, 1985, **132**, pp. 134–143

[6] RUMELHART, D. E., McCLELLAND, J. L., and the PDP Research Group: 'Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations' (MIT Press/Bradford Books, Cambridge, Massachusetts, 1986)

[7] MILLER, W. T., III, HEWES, R. P., GLANZ, F. H., and KRAFT, L. G., III: 'Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller', *IEEE Trans.*, 1990, **RA-6**, (1), pp. 1–9

[8] NARENDRA, K. S., and PARTHASARATHY, K.: 'Identification and control of dynamical systems using neural networks', *IEEE Trans.*, 1990, **NN-1**, (1), pp. 4–27

[9] MILLER, W. T., III, GLANZ, F. H., and KRAFT, L. G., III.: 'Application of a general learning algorithm to the control of robotic manipulators', *Int. J. Robotics Res.*, 1987, **6**, (2), pp. 84–98

[10] TOURASSIS, V. D., and NEUMAN, C. P., 'Properties and structure of dynamic robot models for control engineering applications', *Mech. Mach. Theory*, 1985, **20**, (1), pp. 27–40

[11] LIM, K. Y., and ESLAMI, M.: 'Adaptive controller design for robot manipulator systems using Lyapunov direct method', *IEEE Trans.*, 1985, **AC-30**, (12), pp. 1229–1233

[12] LUH, J. Y. S., WALKER, M. W., and PAUL, R. P.: 'Resolved-acceleration control of mechanical manipulators', *IEEE Trans.*, 1980, **AC-25**, (3), pp. 468–474

[13] CRAIG, J. J.: 'Introduction to robotics: mechanics and control' (Addison-Wesley, Reading, Massachusetts, 1989) 2nd edn.

[14] PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., and VETTERLING, W. T.: 'Numerical recipes in C: the art of scientific computing' (Cambridge University Press, Cambridge, UK, 1988)

The paper was first received on 2 September 1991 and in revised form on 23 March 1992.

The authors are with the Department of Mechanical and Production Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 0511.