Performance of Two Neuro Controllers For Robot Path Planning Control

Shamsudin H.M. Amin, Otman Musa Ahtiwash
Faculty of Electrical Engineering
Universiti Teknologi Malaysia
Locked Bag 791, Skudai
80990 Johor Bahru, Johor, Malaysia
Email: sham@fkeserv.fke.utm.my

Abstract - **In this paper, two neuro-controllers utilizing the back propagation algorithm are investigated for robot path tracking performance: Inverse Neuro-Controller and Neuro-Emulator Neuro-Controller schemes. For a given task of moving a robot from a rest position to a final specified position in a minimum-time, the resulting position and velocity profiles for the investigated neuro-control models showed that, the manipulator could be moved smoothly and accurately. The tracking performance and accuracy are investigated and compared.**

I. INTRODUCTION

Neural Networks have emerged rapidly in the last few years as a possible candidate to solve real-time problems because of their parallel computation structure and learning ability. The neural network parallel structure can ease the real-time problem of high computation requirement. However, through on-line or off-line learning, neural network controllers can learn the system characteristics from the input/output data, and hence find a way to control it.

In this paper we will make use of these advantages of neural networks to solve for the path control problem. Here a neural network controller is designed on a multi-layered network, for which the adaptation of system changes could be accommodated via the back propagation of errors through different layers.

We have reported earlier the investigations in obtaining minimum-time path planning [1] and a neural networks approach for intelligent path planning [2]. Here we are reporting further developments in investigation of use of neuro controllers: Inverse Neuro-Controller and Neuro-Emulator Neuro-Controller approaches.

The Neuro-Emulator Neuro-Controller scheme can be regarded as an integrated and modified model of the inverse neuro-controller scheme. This approach was first proposed and then well defined by other researchers [3][4][5].

II. PATH PLANNING CONTROL PROBLEM

The standard task of robot path control is to map a desired trajectory $q_d(t)$ onto n joint torque functions $\tau_d(t)$ for the corresponding n joint of a manipulator, such that its end-effector moving along its actual trajectory $q_a(t)$ remains as close as possible to $q_d(t)$. There are three fundamental levels in path control to accomplish this movement: task planning, trajectory path planning and path motion control.

* In *task-planning level*, the planners manage and coordinate the information (e.g., via-points, obstacles, end-effector position and orientation, ... etc.,) of the job to be performed, which involves providing the solution of inverse kinematics.
* The *trajectory planning level*, in which, given the initial, target coordinates as well as the appropriate constraints, the sequence of the points and the desired trajectory through which the end-effector must pass are to be found.
* Given such a trajectory (objective position, velocity and acceleration), the *path control level*, consists of finding the necessary n joint torques to move the end-effector along this desired trajectory, even in presence of unexpected load changes.

The path control problem focuses on the computation of the actuating joint torques/forces that are required to produce the reference trajectories, which involves solving of the robot dynamics problem. The dynamic model which describe the motion of n joint robot, that relates the actuating joint torques $\tau_i(t)$ with the joint positions , velocities and accelerations ($q_i(t)$, $\dot{q}_i(t)$, $\ddot{q}_i(t)$) is as given below:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F(q,\dot{q}) \quad (1)$$

Where q is (nx1) vector of joint displacements, $\dot{q}$ is the (nx1) vector of the joint velocities, $\ddot{q}$ is the (n x 1) vector of the joint accelerations, τ is the (n x 1) vector of applied torques, $M(q)$ is the (n x n) symmetric inertia matrix, $C(q,\dot{q})$ is the (n x 1) vector of centripetal and coriolis forces, $G(q)$ is the (n x 1) vector of gravitational forces, $F(q,\dot{q})$ is the uncertain vector of the dynamics including the friction and any other disturbances, n the number of degrees of freedom of the robot.

These equations are typically very complex, highly coupled and non-linear. However, most commercial manipulators are equipped with controllers that ignore these non-linear robot dynamics. These simplified controllers may fail to characterize the complex joint dynamics and coupling, resulting in oscillations or overshoot of the end effector. As a result, the design of an ideal controller for such a system is currently one of the most challenging tasks to achieve a better performance while taking into account the dynamics of the robot. Commonly, there are two barriers to the successful implementation of the conventional controllers[6].

i. The computation of the complex non-linear dynamics robot model needs to be done in real time, typically less than 10 ms, and this is hard to achieve with present single-processor technology.

0-7803-2775-6/96 $4.00 © 1996 IEEE

ii. The parameters in the dynamics model of the robot must be known precisely.

The neural network for learning the robot dynamics can be regarded as an example of the autonomous driving torque generator. i.e. the neural network model generates the necessary driving torques in the robot joints as a nonlinear mapping of the robot desired joint displacement, joint velocities and joint accelerations:

$$\tau_i = f(W_{ji}\ ,\ q_d\ ,\ \dot{q}_d\ ,\ \ddot{q}_d) \qquad (2)$$

where, τ is the (n x 1) vector of joint driving torques, W_{ji} are the adaptive weight matrices between the network layers, and $f(.)$ is the non-linear mapping sigmoidal activation function. In the following sections, we will discuss two different approaches suggested to learn the dynamics of the robot model in order to control its movement.

III. INVERSE NEURO-CONTROLLER SCHEME

In this scheme the neural network is directly controlling the robot, where the neural network is placed at the input path of the controlled system acting as a feedforward controller. This is regarded as the specialized learning architecture as been shown in Fig. 1, where the neural network is trained on-line to learn the inverse model of the robot by back-propagation of the performance error ε_p [7]. However, it is necessary at the beginning to train the neural network *off-line* using the generalized learning architecture. Once it converges (i.e. learns the true inverse of the robot model), it can be put in the feedforward path and continue on-line tuning and adapting to changes in the environment and within the system [8].

A. Off-line Learning

The off-line learning is regarded as the first stage of training the network to learn the inverse of the robot dynamics. As shown in Fig. 2, a set of commanded torques, denoted as (τ_d), are used to drive the robot producing a set of resulting actual trajectories (q_a). The network receives the actual trajectory (q_a) as input and produces a set of actual motor torques, denoted as (τ_a). The goal of the generalized learning is to minimize the errors between (τ_a) and (τ_d) in the least square sense using the back-propagation algorithm. After the network is fully trained, if any new applied input (q) is sufficiently close to the reference trajectory (q_d), then the controller should be able to reproduce a proper torque τ, making the actual movement closely following the desired trajectory. Then the network will be used as a feedforward controller for the robot.

B. On-line Learning

In order to improve the performance of the neural networks which have been trained using the off-line general learning method, we incorporate the specialized learning stage into the

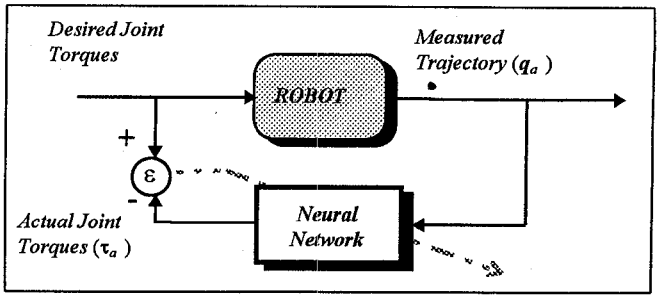

Fig. 1 Generalized Learning Configuration

control system. The neural network is trained based on the regions of interest, where the desired trajectory (q_d) is applied directly to the neural input layers and outputs the appropriate torque commands (τ_d) to drive the robot resulting a movement trajectory (q_a). The values of the desired trajectory and the actual movement trajectory are compared yielding a performance error (ε_p). The performance error, (q_d - q_a) is then back-propagated through the network at every sample. In order to accomplish the on-line learning, the sign of the robot's partial derivative $\frac{\partial q_a}{\partial \tau}$ was used to approximate the robot Jacobian according to [9]. Thus, when the operating points of the system change or when new training patterns is added, it should be adequate to use specialized learning to fine tune the system.

IV. NEURO-EMULATOR NEURO-CONTROLLER

As shown in Fig. 3, this architecture has two Neural networks used to control the plant. The first is used as an emulator to emulate the plant behavior and trained to learn the plant forward dynamics, while the other is used as a controller trained to learn the plant inverse dynamic. Thus, this approach allows more accurate training on-line for the neuro-controller as the performance error (ε_p) is backpropagated through the emulator at every sample.

Using the back-propagation algorithm, the neuro-controller is trained to learn the inverse model of the plant in the same way firstly, off-line, and then on-line. On the other hand, the emulator is first trained off-line to learn the plant forward model by injecting a recorded input data together with some delayed output data as the input patterns and the corresponding outputs as the target patterns. Then the emulator is embodied in the system to continue on-line training.

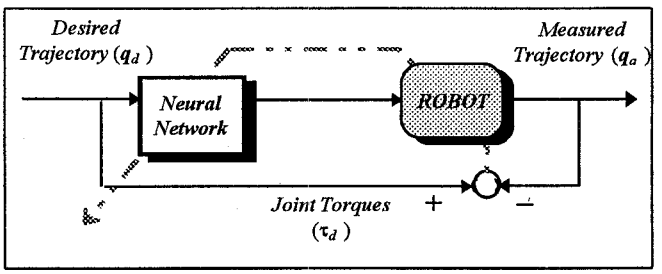

Fig. 2 Specialized Learning Configuration

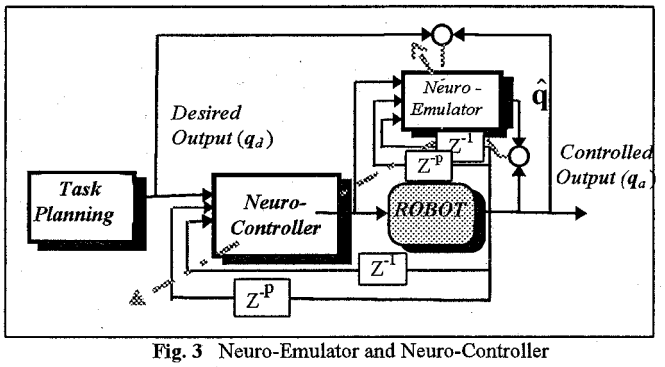

Fig. 3 Neuro-Emulator and Neuro-Controller

A. Emulator off-line Learning

At the first stage of learning, the emulator is trained off-line to learn the forward model of the robot such that it would approximate its behavior. The training is implemented by applying the same inputs of the robot together with some delayed outputs. To accomplish this stage of training the back-propagation algorithm is again used and the weights of the emulator are adjusted using (3) and (4). For the output layer

$$\Delta W_{kj}(t) = \eta\, \delta_k\, y_j + \alpha\, \Delta W_{ji}(t-1) \quad , \qquad (3)$$

and, for the hidden layer

$$\Delta W_{ji}(t) = \eta\, \delta_j\, x_i + \alpha\, \Delta W_{ji}(t-1) \quad , \qquad (4)$$

where, α is usually a positive number called the momentum parameter.

A. On-line Learning

At the second stage of learning, the on-line learning method is adopted to improve the performance of the system being controlled [5]. The performance error

$$\varepsilon_p = \frac{1}{2}(q_d - q_a)^2 \quad , \qquad (5)$$

is used as the error signal to update the neuro-controller weights after being back-propagated through the emulator, where

$$\delta_k = (q_d - q_a) \quad , \qquad (6)$$

and, δ_k in this case is the error between the hidden and output layers of the emulator . Then the errors between the hidden and the input layers are adjusted in the same way described for the back-propagation algorithm.

In order to improve the operation of the emulator itself, the emulator weights are updated on-line by back-propagating the errors resulting by comparing the emulator outputs and the robot model outputs at every sample.

$$\varepsilon_1 = \frac{1}{2}(q_a - \hat{q})^2 \quad , \qquad (7)$$

where, $\hat{q}$ is the emulator output.

Two advantages will be obtained using this approach: Firstly, the Jacobian of the partial derivatives of the plant can be approximated by the neuro-emulator instead of using their signs because the neuro-emulator is acting as a plant identifier which means that the change in the plant output corresponding to the plant input can be detected. Secondly, it allows for more accurate training of the neuro-controller on-line as the performance error (ε_p) is propagated back through the emulator at every sample.

V. CASE STUDIES

To illustrate the capability of the techniques presented in preceding sections, we performed the minimum-time path planning control on a model of PUMA like robot. The first 3 joints (Arm joints) of a 6 joint PUMA manipulator are responsible for positioning the end-effector, while the last three joints (Wrist joints) were used for the end-effector orientation. Thus, in our simulations we performed the path planning control for only the first 3 joints, where the payload and the mass of the wrist joints were presented as an effective load at the end of the third link [10].

A. Pre-training of the Neural Networks

In this simulation study we use a 3-layer (with one hidden layer) back-propagation neural network, with a configuration of (3 × *n*) neurons in the input layer, and *n* output neurons at the output layer yielding the required joint torques, where 3 represents the input desired trajectory parameters (q_d , $\dot{q}_d$, $\ddot{q}_d$) for each joint, and *n* is the number of the robot *d.o.f.* The number of the hidden layer neurons is determined by simulation and experience rather than any fixed rule, therefore we use the trial and error procedure by varying the number of the hidden neurons and check for the validity of the obtained results. It was found that as the number of the hidden neurons increased the neural network accuracy improved, but in contrast it caused slower learning. Also, due to the large number of the hidden neurons, it may drive the neural network to memorization instead of generalization [11][12]. Through experiments we found that a hidden layer with 24 neurons is optimum for our application in terms of speed and accuracy. Thus, the structure of our proposed neural network to learn the inverse of the robot model has (9-24-3) (number of neurons for input, hidden and output layers respectively). The non-linear activation function of the hidden neurons chosen as a hyperbolic tangent function $f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$, while for the output layers was chosen as a linear function: $f(x) = x$. Before starting the off-line training all the weights were randomly initialized at the range of ± 0.5 .

B. Off-line Learning

Before implementing the robot path control, the neural network is trained in an off-line way to learn the inverse dynamics using the back-propagation. With a fixed

momentum coefficient of 0.8 and a learning rate coefficient initially chosen at 0.001 to speed up the learning and then reduced to 0.0001 to avoid the system instability. The neural network weights are updated in accordance to (3) and (4).

The training patterns consist of uniformly distributed values of joint parameters (positions, velocities and accelerations) as training inputs, and their corresponding joint torques as training outputs. As the training proceeds the neural network starts to realize the robot dynamics and provides the accurate mapping, and the convergence error decreases gradually as the number of iterations increases as can be observed in Fig. 4. The performance of the training procedure was checked after 50,000 iterations and the results obtained by the neural network is shown in Fig. 5, where the improvement of learning the plant inverse is clearly observed. It can be observed that the neural network performed fairly well in learning the robot inverse dynamics model.

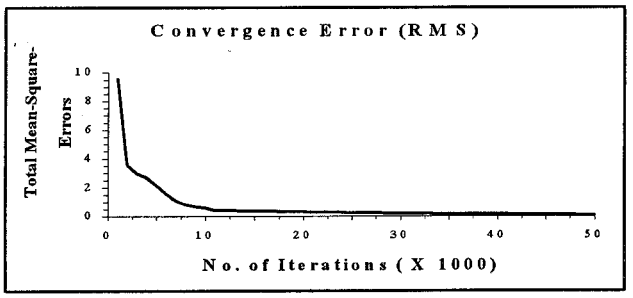

Fig. 4 : Covergence of Error for the first 50,000 Training Iterations

C. On-line Learning

After the neural networks have learned the true inverse of the dynamic model of the robot, these trained neural networks are employed to act as neuro-controllers in the suggested schemes as described earlier, to perform on-line learning in the regions of interest and to adapt for any changes during controlling the robotic motion in tracking the desired trajectory. Two case studies were carried out to demonstrate the trajectory tracking capability of the neuro-controllers to follow the predefined trajectories.

1) Case I : The manipulator was required to follow a minimum-time trajectory, which started from initial rest position (initial velocity and acceleration = 0) to a fully stop condition at the end of the trajectory (final velocity and acceleration = 0). The computed torque inputs generated by the neuro-controllers were used to drive the robot arm to follow and track the desired minimum-time trajectory. The simulated manipulator response in terms of joint positions and velocities are shown for the different proposed neuro-control schemes.

* Fig. 6 shows the position tracking responses for the three joints using the Inverse Neuro-Controller. It was observed that there are significant errors during the initial path stages which is due to the reason that the neuro-controller was still trying to adapt itself initially. As time progress these errors rapidly disappeared.
* The position tracking responses for the Neuro-emulator Neuro-controller are shown in Fig. 7. It is interesting to note that these results demonstrate smaller errors than the previous model because of the presence of the neuro-emulator which identify the exact forward robot model which provide the neuro-controller with the performance error at every sampling time.

A summary of the calculated maximum errors for the three proposed architectures in terms of position and velocity tracking responses to follow a proposed minimum-time trajectory is given in Table 1.

TABLE 1
TRACKING ERRORS FOR CASE I

CASE (I): Maximum Position and Velocity Errors				
MODEL	*ERROR*	*Joint* **1**	*Joint* **2**	*Joint* **3**
Inverse Neuro-Controller	Position Error(rad)	0.123	0.181	0.237
	Velocity Error(rad/s)	0.098	0.017	0.213
Neuro-Emulator Neuro-Controlle	Position Error (rad)	0.102	0.079	0.192
	Velocity Error (rad/s)	0.075	0.034	0.025

In Table 1, it can be observed that the tracking errors for all the schemes are relatively small. It appears that the Neuro-Emulator Neuro-Controller model gives the best results.

2) Case II : In this case of study, the first three joints of the robot manipulator are required to move from an initial position of { $q_i \in$(-0.8, -1.5, -0.5) radians } to a final position of { $q_i \in$(1.0, 0.2, 1.2) radians }. The initial and final velocities and accelerations are all zero [13]. The three neuro-controller architectures are simulated to follow and track this desired trajectory. The performance of the different architectures is noticed to be improved for this short trajectory which needed only a duration of 2 seconds to be accomplished.

* The position and velocity tracking profiles of the Inverse Neuro-Controller architecture are shown in Fig. 8. It can be observed that the performance of the neuro-controller improved and was able to track the desired trajectory very smoothly, which represents the efficiency of this model to track different desired trajectories.
* Fig. 9 illustrates the position and velocity tracking of the Neuro-emulator Neuro-Controller architecture. It can be observed that the smoothness and accuracy of tracking of the desired trajectory is almost the same as that presented by the previous architecture.

Table 2 illustrates the maximum errors values for the joint positions and velocities for the three different schemes. From Table 2 , it is noticed that the Neuro-emulator Neuro-Controller model and the Inverse Neuro-Controller model

demonstrate very close performance, and give the best tracking results for this short trajectory.

TABLE 2
TRACKING ERRORS FOR CASE II

CASE (II): Maximum Position and Velocity Errors				
MODEL	*ERROR*	*Joint* 1	*Joint* 2	*Joint* 3
Inverse Neuro-Controller	Position Error (rad)	0.016	0.014	0.014
	Velocity Error (rad/s)	0.101	0.029	0.036
Neuro-Emulator Neuro-Controller	Position Error (rad)	0.015	0.014	0.014
	Velocity Error (rad/s)	0.094	0.028	0.036

VI. CONCLUSIONS

The proposed neural networks were trained to learn the inverse dynamic model of the robot in such a way that they will be able to produce the necessary actuating torques to track a specified minimum-time trajectory. Two schemes namely Inverse Neuro-Controller and Neuro-Emulator Neuro-Controller model, utilizing the back-propagation algorithm have been discussed.

For a given task of moving a robot from a rest position to a final specified position in a minimum-time, the resulting position and velocity profiles for the investigated neuro-control models showed that using neural networks, the manipulator could be moved smoothly and accurately, which clearly exhibited the capability of the neural networks towards the tracking of a desired motion trajectory. The tracking performance and accuracy are investigated and compared. The results indicated that the Neuro-Emulator Neuro-Controller model gives the best tracking performance because of the presence of the neuro-emulator which is used to emulate the forward behavior of the robot model.

VII. REFERENCES

[1] O.M. Ahtiwash, S.H.M. Amin, "Intelligent Robot Path Trajectory Control." in *Proceedings of International Conference on Robotics, Vision And Parallel Processing For Industrial Automation.* ROVPIA'94, pp.420-426

[2] O.M. Ahtiwash, S.H.M. Amin, "A Neural Network Approach For Robot Path Trajectory Control." *Proceedings of First Asian Control Conference* ASCC-94, vol. 2, pp. 793-796.

[3] P.J. Werbos, P. J. (1990). "Back Propagation through time : What it does and how to do it ?." *IEEE* Vol.78, no.10; 1550-1560.

[4] D. Nguyen, B. Widrow, "The truck backer-upper : an example of self-learning in neural networks" *Proceedings of International Joint Conference on Neural Networks* , 1989, Vol. 1, pp.357-353.

[5] J. Tanomaru, S. Omatu, "Process Control by On-Line Trained Neural Controllers." *IEEE Trans. on Industrial Electronics,* vol. 39, pp. 511-521.

[6] A.N. Poo, M.H. Ang Jr., C.L. Teo, Q. Li, "Performance Of A Neuro-Model-Based Robot Controlller: Adaptapility And Noise Rejection." *Intelligent Systems Engineering*, Autumn 1992, pp.50-62.

[7] D. Psaltis, A. Sideris, A. Yamamura, "A multilayered neural network controller." *IEEE Control System Magazine* , 1988,vol. 8, no.3,pp.17-21.

[8] S. Kung, J. Hwang, "Neural Network Architecture for Robotic Applications", *IEEE Trans on Robotics and Automation,* 1989, vol.5. no.5, pp.641-653.

[9] M. Saerens, A. Soquet, J.M. Renders, H. Bersini, "Preliminary comparisons between a neural adaptive controller and a model adaptive reference controller." *Proceedings of the International Conference on Neural Networks*, 1990,

[10] K. Jouaneh Musa, D.A. Dornfeld, M. Tomizuka, "Trajectory planning for coordinated motion of a robot and a positioning table: Part 2 - Optimal trajectory specification", *IEEE Transactions on Robotics and Automation.,* 1990, vol.6, no.6; 746-751.

[11] M. Jamshidi, B. Horne, N. Vadice, "A Neural Network Based Controller for a Two Link Robot" Proceedings of 1990 IEEE Proc. on Decision and Control, pp. 3256-3257.

[12] J.M. Zurada, *Introduction to Artificial Neural Systems,* Info Access & distribution Pte Ltd, Singapore, 1992

[13] J.H.S. Osman, *Decentralized and Hierarchical Control of Robot Manipulators,* 1991, Ph.D. Thesis, City University of London

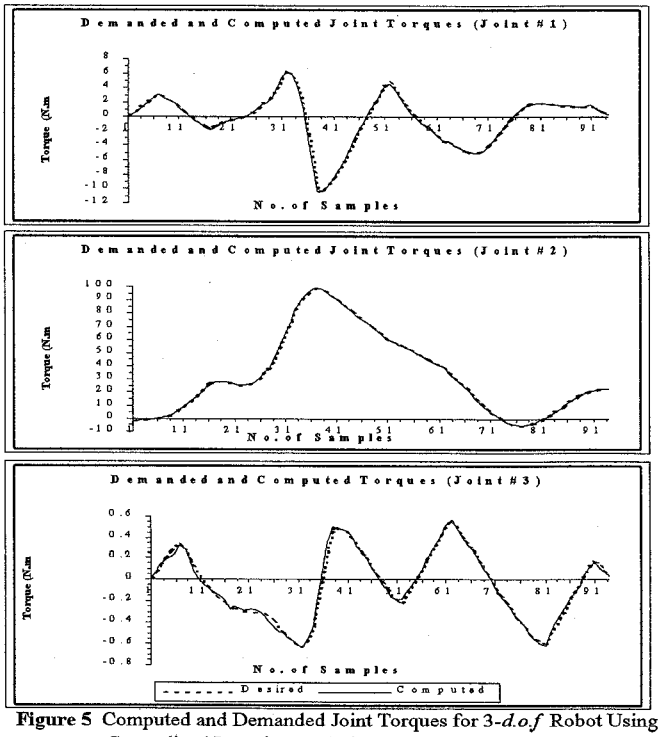

Figure 5 Computed and Demanded Joint Torques for 3-*d.o.f* Robot Using Generalized Learning Technique (Inverse Robot Model)

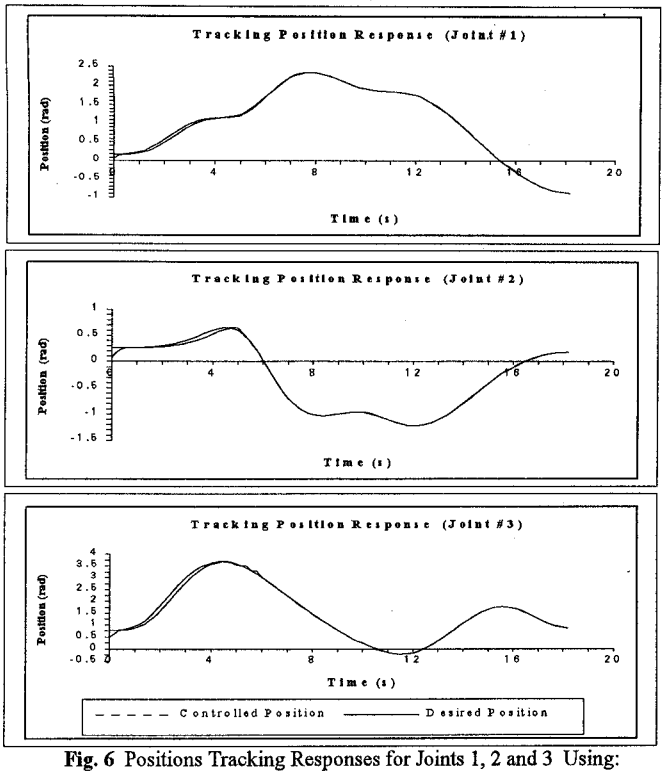

Fig. 6 Positions Tracking Responses for Joints 1, 2 and 3 Using: **(Inverse Neuro-Controller Model)**

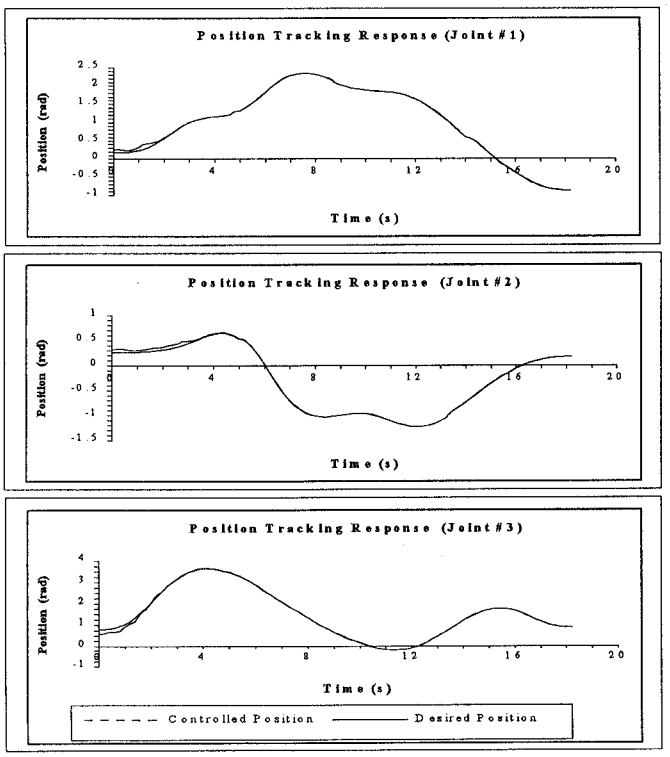

Fig. 7 Position Tracking Response for Joints 1,2 and 3 Using : (Neuro-Controller Neuro-Emulator Model) CASE No. (I)

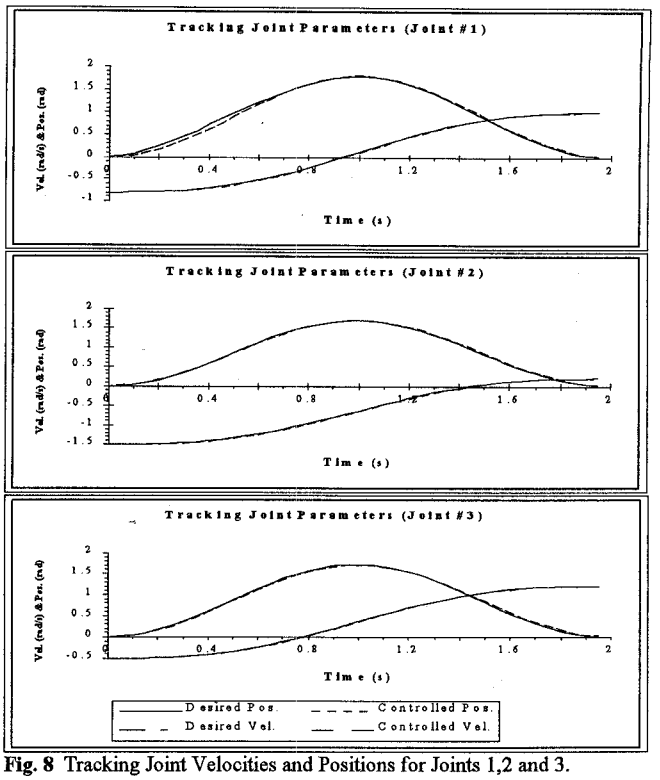

Fig. 8 Tracking Joint Velocities and Positions for Joints 1,2 and 3. **(Inverse Neuro-Controller Model) CASE No. (II)**

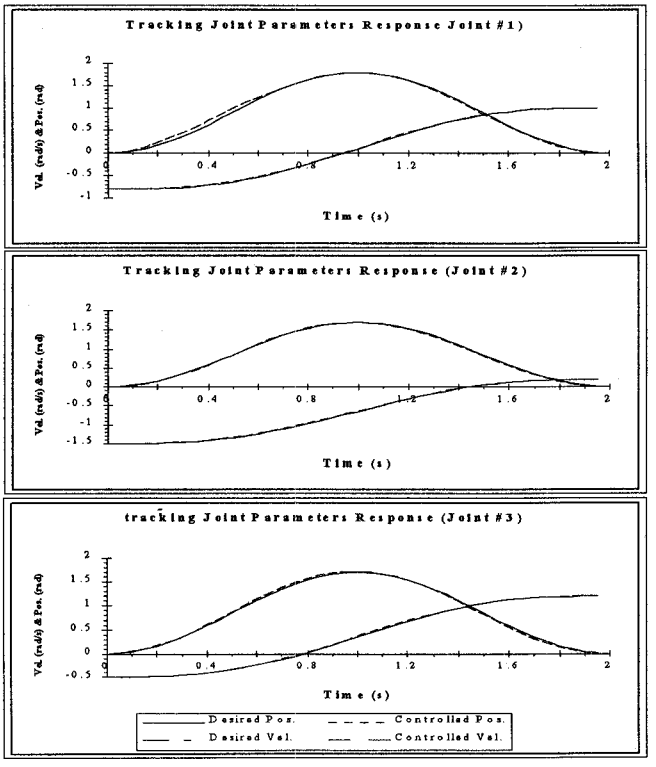

Fig. 9 Tracking Joint Velocities and Positions for Joints 1,2 and 3. (Neuro-Controller Neuro-Emulator Model) CASE No.(II)