A NEURAL NETWORK ARCHITECTURE FOR EMULATING FORWARD DYNAMICS OF A ROBOT MANIPULATOR

Turhan Tunalı [1] Mehmet Kuntalp [2] Bahadır Şamlı [1]

1 Dept. of Computer Engineering, Ege University, Bornova, Izmir 35100, Türkiye

2 Dept. of Electronic and Communication Engineering, Dokuz Eylül University, Bornova, Izmir 35100, Türkiye

Abstract

A neural network architecture for emulating forward dynamics of a simulated three degree of freedom revolute joint robot manipulator among a predefined trajectory is developed. The performance is investigated and results are reported. The potential use of the emulator for robust control purposes is discussed.

1. INTRODUCTION

It is well known that the technique of error backpropagation attracted many researchers and incereased the popularity of artificial neural networks. As in most other neural network studies, application of neural networks to control systems involves the use of error backpropagation technique [1,2,3]. In these methods, a feedforward neural network is trained to behave like the plant which is to be controlled. Afterwords, the neurocontroller is trained. In other words, the feedforward neural network acts as an "emulator" and it learns to identify the dynamical characteristics of the plant. The training of the emulator is analogous to "plant identification" in control theory however there are two basic advantages. Firstly, there is no need to provide any information about the model. That is, emulation is done automatically. Secondly, the neural emulator is able to model the nonlinear plants including robot manipulators.

The importance of providing a good emulator for neurocontroller architectures lies under the fact that the neurocontroller cannot be trained by using error backpropagation algorithm without an emulator. The output values that can be measured are the ones of the plant, not of the neurocontroller and the error signal at the output can not propagate backward through the plant which is not a neural network; so the backpropagation algorithm in which the error must propagate backward from the output to the input can not be used. To overcome this problem, the emulator is used in place of plant during the training of neurocontroller.

In this study, a neural network architecture for emulating forward dynamics of a simulated three degree of freedom revolute joint robot manipulator among a predefined trajectory is developed. In the next section, the simulated forward dynamics of the robot arm used in training of the emulator will be given and the trajectory used will be specified. In the third section, details about the training process and the architecture will be discussed. Finally in the last section simulation results about the performance of the emulator will be given and the ability of the developed emulator in operating on different trajectories will be discussed.

2. ROBOT ARM FORWARD DYNAMICS SIMULATION

Using Lagrange-Euler formulation the dynamics of a robot arm can be written as

$$A(\Theta)\ddot{\Theta} + H(\Theta, \dot{\Theta}) + G(\Theta) = \tau \qquad (1)$$

where $\Theta \in R^n$ is the joint angle vector, $\dot{\Theta} \in R^n$ is the joint angular velocity vector, $\ddot{\Theta} \in R^n$ is the joint angular acceleration vector, n is the number of joints, $\tau \in R^n$ is the motor torque vector. $A(\Theta) \in R^n \times R^n$ is the generalized inertia matrix, $H(\Theta, \dot{\Theta}) \in R^n$ is the nonlinear term containing coriolis and centipital factors, $G(\Theta)$ is the term containing gravity factors [4]. The link structures used in this study are circular cylindrical shell, conical shell and a uniform slender rod as given in [4] for a Microbot arm. The values of parameters used are m_1= 3kg,m_2= 2kg , m_3= 1 kg where m_i is the ith joint mass, h= 20 cm e= 30 cm f= 30 cm where, h, e, f are joint lengths respectively, r= 5 cm is the link radius. Θ and $\dot{\Theta}$ are the outputs and τ is the input and n=3.

We have used

$$x(t) = a + bt + ct^2 + dt^3 \qquad (2)$$

to generate desired trajectory in base coordinates which involves going from one point to another in 2.5 sec. The initial and final points are assumed to be 80 cm apart from each other. The inverse kinematics package written for the robot generates $\Theta_d(t)$, $\dot{\Theta}_d(t)$ and $\ddot{\Theta}_d(t)$ which realizes (2) every 10 msec which is the sampling period T. The robot is assumed to be at rest at the initial and final configurations.

We have used computed torque technique [5] to generate the torque values which make robot track the desired trajectory. The robot is assumed to be on the desired trajectory initially. The output data $\Theta(t)$ and $\dot{\Theta}(t)$ is generated by using Runge-Kutta integration procedure of fourth order. As a result we have generated data for 250 sampling intervals which contain $\Theta(kT)$, $\dot{\Theta}(kT)$, $\tau(kT)$.

3. EMULATOR ARCHITECTURE AND TRAINING SCHEME

The neural network is chosen to be a feedforward one which consists of an input layer, two hidden layers and an output layer. The number of processing elements in the output layer is chosen to be six which is equal to the number of outputs of the robot arm. The input layer has nine elements among which three are the input torques and six are the initial states. Figure 1 depicts the inputs and outputs of the neural emulator. In the first hidden layer

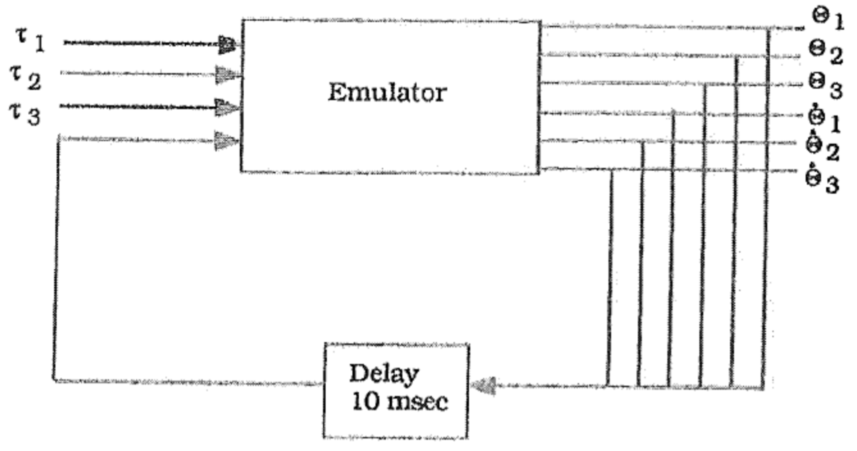

Figure 1: The Inputs and Outputs of the Neural Emulator

27 processing elements are used. The number of processing elements in the second hidden layer is chosen as that of the output layer, namely six. Thus, as a total, emulator network contains 48 processing elements and 441 connections among them.

Error backpropagation algorithm is used to train the emulator. Figure 2 depicts the training scheme of the emulator.

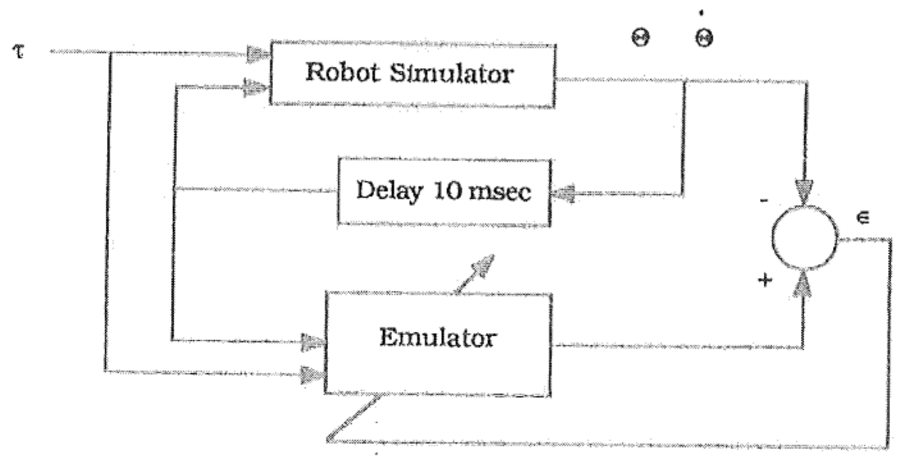

Figure 2: Training Scheme

The environment used is Neural Works Professional II on a PC. The scale factors are chosen to be 0.25, 1.00, 1.00 and 2.50 respectively. All of the transfer functions are chosen to be hyperbolic tangent except the ones used in the input layer for which linear transfer functions are used.

The order of the data generated in section 2 which contains 250 sampling interval input and outputs is randomized to prevent network from diverging. The initial connection weights are also randomized between -0.1 and +0.1 due to same reason. To improve the convergence in training process, the values of the learning coefficients are decreased several times during training. Table 3 shows this variation.

Training up to / Learning coefficient	10000	15000	20000	30000	40000	50000	60000	70000
1	0.900	0.750	0.600	0.300	0.225	0.150	0.135	0.105
2	0.600	0.500	0.400	0.200	0.150	0.100	0.090	0.070
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 3: Change in Learning Coefficients

4. PERFORMANCE OF THE EMULATOR AND DISCUSSION

To measure the performance of the emulator, a program is written in C language to evaluate the errors at the outputs. After training the network 70000 times, it has been observed that the rms positional error is reduced to 0.23 radians approximately. It has also been observed that if the robot is initially on desired trajectory and computed torque method is used on simulated dynamics alone, comparable rms error occurs even if T is decreased. This indicates that the rms error measured at the output of the emulator is mainly because of numerical errors of PC environment. Table 4 indicates the mean, rms and absolute extremum errors of joint positions and velocities of the emulator.

	mean error	rms error	absolute extremum error
θ_1	-0.0023	0.0264	0.0575
θ_2	0.0059	0.0267	0.0640
θ_3	0.0027	0.0162	0.0267
$\dot{\theta}_1$	0.0011	0.0316	0.0866
$\dot{\theta}_2$	-0.0072	0.0241	0.0532
$\dot{\theta}_3$	0.0089	0.0174	0.0406

Tablo 4: Mean, rms and absolute extremum errors (in radians and radian/sec)

The next step was to determine how succesfully the network emulates the manipulator on trajectories which are not thaught. In other words, how succesfully the emulator generalizes? The emulator is tested as follows: The trajectory for which the emulator was trained is taken as a reference. Then to determine the generalization range, various trajectories which are x cm apart from the reference trajectory were created and network is tested with these new trajectories. Namely, torque values which correspond to these trajectories are given as input and the outputs of the simulated robot and emulator are compared and rms errors are examined. Then x is increased and the measurements are repeated.

We have concluded that emulator performance decreases when $x>1.5$ cm. In other words, emulator has learned the trajectories which are within 1.5 cm of the original one. This is good enough to use the emulator for robust control purposes in which the magnitude of error never reaches to these values. Tables 5 indicates the mean, rms and absolute extremum errors measured for trajectories which are 1.5 apart from the original one.

	mean error	rms error	absolute extremum error
θ_1	-0.010	0.057	0.101
θ_2	0.072	0.036	0.110
θ_3	0.098	0.046	0.111
$\dot{\theta}_1$	0.002	0.073	0.178
$\dot{\theta}_2$	-0.006	0.082	0.184
$\dot{\theta}_3$	0.029	0.087	0.215

Table 5: Mean, rms and absolute extremum errors (in radians and radian/sec) for trajectories 1.5 apart from the original one.

REFERENCES :

[1] Psaltis, D., Sideris, A. and Yamamura A. (1988). "A Multilayered Neural Network Controller" IEEE Control Systems Magazine, April.

[2] Kawato, M., Ono, Y., Isobe, M. and Suzuki, R. (1987). "Hierarchical Neural Network Mode for Voluntary Movement with Application to Robotics" Proceedings of the IEEE International Conference on Neural Networks.

[3] Gues, A., Eilbert, J.L. and Kam, M. (1987). "Neural Network Architecture for Control" Proceedings of the IEEE International Conference on Neural Network.

[4] Wolowich, W.A. (1987). "Robotics: Basic Analysis and Design". Holt, Rinehart and Winston 1987.

[5] Lee, C.S.G., Mudge, T.N. and Turney, J.L. (1982). "A Hierarchical Control Structure Using Special Purpose Processors For the Control of Robot Arms" Proc. of the Pattern Recognition and Image Proccessing Conference.