A NEURAL NETWORK ADAPTIVE CONTROL SCHEME FOR ROBOT MANIPULATORS

Q.-H. Max Meng

Department of Electrical Engineering
Lakehead University
Thunder Bay, Ontario
Canada P7B 5E1

W.-S. Lu

Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C.
Canada V8W 3P6

ABSTRACT

In this paper, a neural network adaptive control scheme for robot manipulators is proposed which consists of one Adaline network to identify structured system dynamics and another one to compensate for both structured and unstructured dynamic uncertainties. The former is trained off-line using a LMS type algorithm while the latter uses an on-line stable weight updating mechanism determined using Laypunov theory. Since Adaline nets match robot regressor dynamics perfectly, the training processes of the resulting simple neural networks are computationally efficient and the proposed adaptive control scheme has very high potential in real-time applications. The proposed control scheme is finally illustrated through simulation and comparison studies.

1. INTRODUCTION

Interest in applying neural networks to control and identification of linear and nonlinear systems, such as robotic systems, has been growing steadily [1-6]. This is mainly due to the fact that the application of neural networks promises a high computational efficiency provided by the massive parallelism and the demonstrated ability of neural networks to learn and to construct complex and nonlinear mappings through effective training with a great degree of robustness, or fault tolerance due to the distributed representation.

Majority of the previously reported systems [1-5] use a neural network in feed-forward loop which is trained to learn the inverse dynamics of the controlled system through a back-propagation algorithm. In other systems, a neural network is used in place of the mappings in the classical design of adaptive control algorithms [1-4,6]. Both types of systems may use either recurrent or feed-forward networks with mltiple layers, with the structure of the network determined based on applications and experience. The resulting systems generally suffer from tedious learning (training) processes and possible instability since the process of setting up the structure of the network and choosing the number of weights in the network is not optimal but rather arbitrary.

In this paper, a neural network adaptive control scheme for robot manipulators is proposed which consists of one Adaline network to identify structured system dynamics and another one to compensate for both structured and unstructured dynamic uncertainties. The former is trained off-line using a LMS type algorithm while the latter uses an on-line stable weight updating mechanism determined using Laypunov theory. Since Adaline nets match robot regressor dynamics perfectly, the training processes of the resulting simple neural networks are computationally efficient and the proposed adaptive control scheme has very high potential in real-time applications. The proposed control scheme is finally illustrated through simulation and comparison studies.

2. ROBOT DYNAMICS

The dynamics of an n degree-of-freedom (d.o.f.) robot manipulator can be described by

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \qquad (1)$$

where $\boldsymbol{\tau}$ is the $n\times 1$ vector of the joint torque; $\mathbf{q}$ is the $n\times 1$ joint displacement vector; $\mathbf{H}(\mathbf{q})$ is the $n\times n$ mass matrix; $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ represent torques due to centrifugal and gravity forces respectively. Two of the important properties of the dynamics (1) are as follows.

Property 1 All robot parameters such as link masses, link lengths, moments of inertia, etc. appear as coefficients of known function of $(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$. By defining each coefficient as a separate parameter, we can write (1) as

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\boldsymbol{\theta} = \boldsymbol{\tau} \qquad (2)$$

where $\mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ is an $n\times r$ matrix of known functions which has been known as the *manipulator regressor*, and $\boldsymbol{\theta}$ is an $r\times 1$ vector of parameters.

Property 2 Matrix $\mathbf{S}(\mathbf{q},\dot{\mathbf{q}}) = \dot{\mathbf{H}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ is skew symmetric, i.e., it satisfies

$$\mathbf{x}^T\mathbf{S}\mathbf{x} = \mathbf{x}^T(\dot{\mathbf{H}} - 2\mathbf{C})\mathbf{x} = 0.$$

3. NEURAL NETWORKS

A neural network is a system with inputs and outputs and is composed of many simple and similar processing elements. The processing elements each have a number of internal parameters called weights. Changing the weights of an element will alter the behavior of the element and, therefore, will also alter the behavior of the network. The goal here is to choose the weights of the network to achieve a desired input/output relationship. This process is known as training the network.

Processing element used in the neural networks of the proposed control system, Adaline [7,5], is shown in Fig. 1. It has an input vector $\mathbf{x} = \{x_i\}$, which contains r components, a single output c, and a weight vector $\mathbf{w} = \{w_i\}$, which also contains r components. The output c equals the sum of inputs multiplied by the weights and then passed through a nonlinear function.

 0-7803-0971-5/93/$3.00

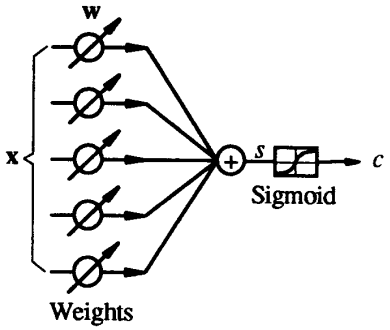

Fig.1 Adaline with sigmoid

$$s(\mathbf{x}) = \sum_{i=1}^{r} w_i x_i \tag{3}$$

$$c(\mathbf{x}) = f(s(\mathbf{x})) \tag{4}$$

The nonlinear function $f(s)$ used here is the sigmoidal function

$$f(s) = \frac{1-e^{-2s}}{1+e^{-2s}} = \tanh(s) \tag{5}$$

With this nonlinearity, the Adaline behaves similar to a linear filter when its output is small and saturates to +1 and -1 as the output magnitude increases.

The robot regressor dynamics (2) can be expressed as

$$\tau_i = \sum_{j=1}^{r} y_{ij}\theta_j \quad \text{for } i = 1,\cdots,n \tag{6}$$

where τ_i is the i-th component of τ, θ_j is the j-th component of θ, and y_{ij} is the (i,j)-th component of $\mathbf{Y}$. Comparing eqn. (6) with eqn. (3), it is clear that Adaline networks match the robot regressor dynamics perfectly. We can use n Adaline networks connected together as a feedforward neural network to emulate robot regressor dynamics, where the vector of known function $\mathbf{y}_i = [y_{i1}\ y_{i2}\cdots y_{ir}]$ matches the neural network input vector $\mathbf{x} = [x_1\ x_2\cdots x_r]$ while θ_j, the vector of known and unknown dynamic parameters, matches the vector of weights of the networks $\mathbf{w}$. Torque τ_i then matches the signal s. And network output c is the mapping of s through the sigmoid.

4. Control and Training Algorithms

Consider the equation of motion (1). Define

$$\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_d - \Lambda\tilde{\mathbf{q}} \tag{7}$$

where $\dot{\mathbf{q}}_d$ is the desired velocity, $\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_d$, and $\Lambda > 0$, and the control torque τ is assigned as

$$\tau = \mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\hat{\theta} - \mathbf{K}\mathbf{s} \tag{8}$$

with $\mathbf{s} = \dot{\mathbf{q}} - \mathbf{q}_r$ and $\hat{\theta}$ is an estimate of θ. If we choose $\Lambda = \lambda\mathbf{I}$, $\mathbf{K} = \lambda\hat{\mathbf{H}}(\mathbf{q})$ and $\hat{\theta} = \bar{\theta} + \Delta\theta$ with $\bar{\theta}$ as an initial estimate and $\Delta\theta$ as a small variation of $\hat{\theta}$, the error dynamics of the system becomes

$$\ddot{\tilde{\mathbf{q}}} + 2\lambda\dot{\tilde{\mathbf{q}}} + \lambda^2\tilde{\mathbf{q}} = \hat{\mathbf{H}}^{-1}\mathbf{Y}\tilde{\theta} \tag{9}$$

where $\tilde{\theta} = \hat{\theta} - \theta$ and $\dot{\tilde{\theta}} = \dot{\hat{\theta}} = \Delta\dot{\theta}$. If the estimate is updated according to

$$\Delta\dot{\theta} = -\mathbf{W}\mathbf{Y}^T(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\mathbf{s}, \quad \mathbf{W} > 0, \tag{10}$$

using property 2 of (1), the Lyapunov function candidate

$$V = \frac{1}{2}[\mathbf{s}^T\mathbf{H}\mathbf{s} + \tilde{\theta}^T\mathbf{W}^{-1}\tilde{\theta}] \tag{11}$$

and its time derivative along trajectories of (1) as

$$\dot{V} = -\mathbf{s}^T(\mathbf{K} - \bar{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}))\mathbf{s} \tag{12}$$

with symmetric matrix $\bar{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})$ as

$$\bar{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}) = [\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{C}^T(\mathbf{q},\dot{\mathbf{q}})]/2$$

guarantee the position and velocity tracking errors converge to zero, which is also clear from eqn. (9) since the estimate update law (10) will drive $\Delta\theta$ to $\theta - \bar{\theta}$ ($\tilde{\theta} = \mathbf{0}$) and therefore $\tilde{\mathbf{q}}$ and $\dot{\tilde{\mathbf{q}}}$ approach to zero.

The first term of control (8) can be expressed as

$$\mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\hat{\theta} = \mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\bar{\theta} + \mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\Delta\theta. \tag{13}$$

The first term of (13) can be implemented by a neural network emulator and the initial estimate $\bar{\theta}$, which is the weight vector of the network, can be obtained through an off-line training process based on a set of trial data of the robot as shown in Fig. 2. The regressor $\mathbf{Y}$ can be obtained by using the formula presented in [9].

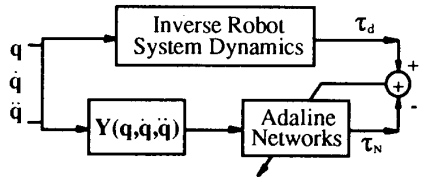

Fig.2 Training the Neural Network Dynamics Emulator

The algorithm used to train neural networks is the least-mean-square (LMS) algorithm of Widrow and Hoff [7,8]. This algorithm converges to a set of weights that minimizes the mean-square error

$$J = \mathrm{E}(\|\tau_d - \tau_N\|^2) \tag{14}$$

where τ_d is the torque vector from the set of trial data of the robot dynamics and τ_N is the output vector of the Adaline networks. By applying gradient descent [7,8], the algorithm to update $\mathbf{w}$ to minimize J turns out to be the following, where $f'(s)$ is the derivative of $f(s)$.

$$w_{i,\text{new}} = w_{i,\text{old}} + 2\mu\delta x_i \tag{15}$$

with

$$\delta = (\tau_{d,i} - \tau_{N,i})f'(s(\mathbf{x})) \tag{16}$$

The designer chooses μ, which affects the speed of convergence and stability of the weights during training. The value d can be thought of as an "equivalent error" and would be equal to the error $\tau_{d,i} - \tau_{N,i}$ if $f(s)$ were the identity function.

The second term of eqn. (13) is implemented using same Adaline networks and the weight vector $\Delta\theta$ is updated on-line according to eqn. (10). Control (8) has been implemented using two Adaline networks. .

5. Simulation and Comparison

A two d.o.f. planar robot shown in Fig. 3 is used as an example system to illustrate the proposed scheme. In this simulation, it is assumed that we have no knowledge of the robot dynamic parameters at all and that the mass of each link is a point mass located at the distal end of the link. By using the formula for regressor dynamics (2) presented in [9], the regressor $\mathbf{Y}$ of the robot is found as

$$\mathbf{Y} = \begin{bmatrix} c_2(\ddot{q}_1 + \ddot{q}_2) & & & & & \\ -s_2\dot{q}_2(\dot{q}_2 + 2\dot{q}_1) & \ddot{q}_1 + \ddot{q}_2 & gc_{12} & gc_1 & \ddot{q}_1 \\ c_2\ddot{q}_1 + s_2\dot{q}_1^2 & \ddot{q}_1 + \ddot{q}_2 & gc_{12} & 0 & 0 \end{bmatrix} \qquad (17)$$

where $c_i = \cos(q_i)$, $s_i = \sin(q_i)$, $c_{12} = \cos(q_1 + q_2)$, and g is the gravitational acceleration constant. Parameter vector θ is

$$\theta^T = \begin{bmatrix} m_2 l_1 l_2 & m_2 l_2^2 & m_2 l_2 & (m_1 + m_2) l_1 & (m_1 + m_2) l_1^2 \end{bmatrix}$$

The true values of the link masses and lengths are m_1=2 kg, m_2=1 kg, l_1=1 m, and l_2=1 m. Therefore the corresponding true parameter vector is

$$\theta^T = \begin{bmatrix} 1 & 1 & 1 & 3 & 3 \end{bmatrix} \qquad (18)$$

In the simulation study, first the training process of Fig.3 is performed where five Adaline networks are connected to form a feedforward neural network to emulate the robot regressor dynamics and a set of small random number is used to start the weight update. After using a set of trial data of the robot dynamics with 200 sampling points for 5 times (200×5 iterations), the estimated weight (parameter) vector $\overline{\theta}$ converges to the following values which are quite accurate

$$\overline{\theta}^T = \begin{bmatrix} 1.0182 & 0.9805 & 1.0021 & 3.0003 & 2.9837 \end{bmatrix}$$

and its convergence profile is depicted in Fig. 4. This set of values is then used as an initial estimate of the parameter vector and the control (8) and (13) with another Adaline network implementing the second term of eqn. (13) is then applied to the robot to track the circle as shown in Fig. 5, which depicts the tracking performance of the neural network compensator. This tracking performance also shows the ability of the proposed neural network compensator in dealing with unstructured dynamics uncertainties since an unmodeled dynamics term, the Coulomb friction term as shown below, is added to the robot dynamics but is not modeled in the control algorithm:

$$\mathbf{f} = \begin{bmatrix} 1.75 \cdot \mathrm{sgn}(\dot{q}_1) & 1.75 \cdot sgn(\dot{q}_2) \end{bmatrix}^T .$$

Fig. 6 shows the tracking errors in joint 1 and 2 by using the neural controller. Fig. 7 shows the profile of the parameter vector variation $\Delta\theta$. Fig. 8 shows the tracking performance of the robot using the modified Slotine-Li adaptive algorithm presented in [9] and its tracking error profile is depicted in Fig. 9. Fig. 10 shows the parameter identification of the adaptive controller, which is not as good as the identification of the neural network emulator since too many parameters (five) are identified and the input signals are not rich enough. Fig. 11 shows the Cartesian space tracking error comparison of the two control methods with the error defined as

$$e(t) = \left\| \mathbf{x}_d(t) - \mathbf{x}(t) \right\|$$

where $\mathbf{x}_d$ is the desired position and $\mathbf{x}$ is the actual position of the robot tip in Cartesian space.

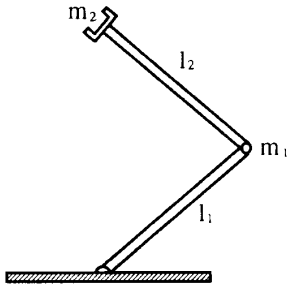

Fig. 3 A Two Degree-of-Freedom Planar Robot

References

[1] W. T. Miller, III, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, The MIT Press, 1990.

[2] G. A. Bekey and K. Y. Goldberg, editors, *Neural Networks in Robotics*, Kluwer Academic Publishers, 1992.

[3] P. J. Antsaklis, "Neural networks for control systems," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 148 and vol. 1, no. 2, pp. 242-243, 1990.

[4] T. Fukuda and T. Shibata, "Theory and applications of neural networks for industrial control systems," *IEEE Trans. Industrial Electronics*, vol. 39, no. 6, pp.472-489, 1992.

[5] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Magzine*, vol. 10, no. 3, pp. 18-23, 1990.

[6] S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4-27, March 1990.

[7] B. Widrow and M. E. Hoff, Jr., "Adaptive Switch Circuits," in *1960 IRE WEST-CON Conv. Record*, Part 4, pp. 96-104, 1960.

[8] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Cliffs, NJ: Prentice-Hall, 1985.

[9] W.-S. Lu and Q.-H. Meng, "Regressor formulation of robot dynamics: computation and applications," to appear in *IEEE Trans. Robotics and Automation*, 1993.

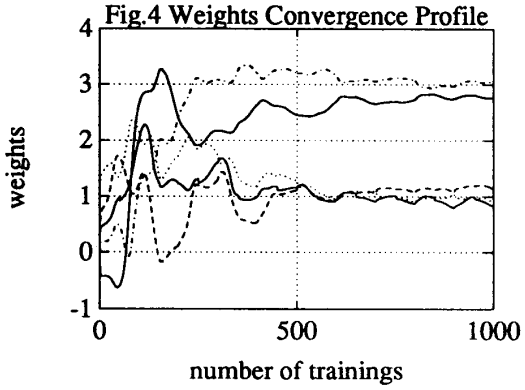
Fig.4 Weights Convergence Profile
weights
number of trainings
4
3
2
1
0
-1
0
500
1000

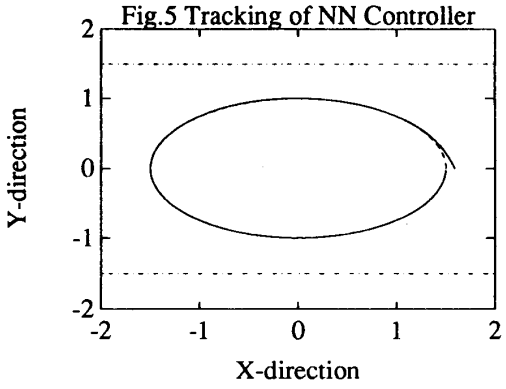
Fig.5 Tracking of NN Controller
Y-direction
X-direction
2
1
0
-1
-2
-2
-1
0
1
2

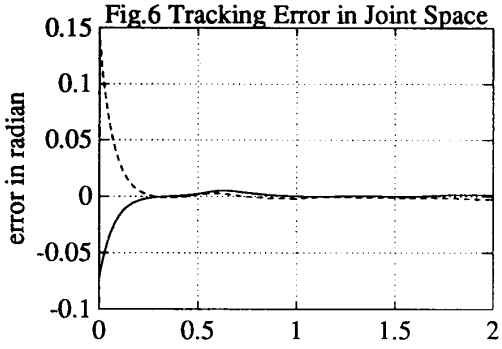
Fig.6 Tracking Error in Joint Space
error in radian
0.15
0.1
0.05
0
-0.05
-0.1
0
0.5
1
1.5
2

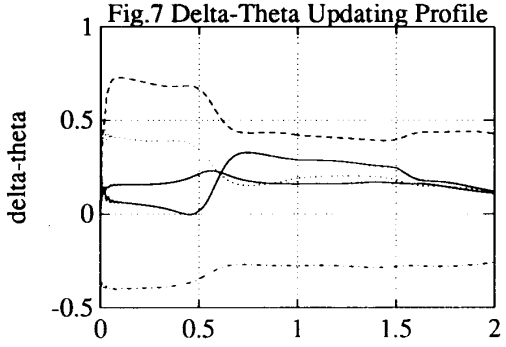
Fig.7 Delta-Theta Updating Profile
delta-theta
1
0.5
0
-0.5
0
0.5
1
1.5
2

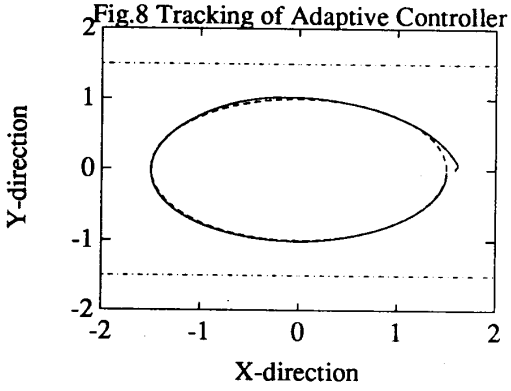
Fig.8 Tracking of Adaptive Controller
Y-direction
X-direction
2
1
0
-1
-2
-2
-1
0
1
2

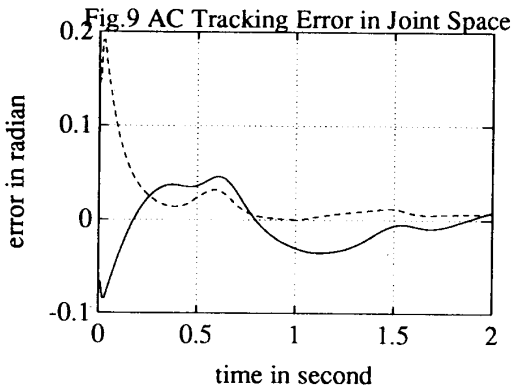
Fig.9 AC Tracking Error in Joint Space
error in radian
time in second
0.2
0.1
0
-0.1
0
0.5
1
1.5
2

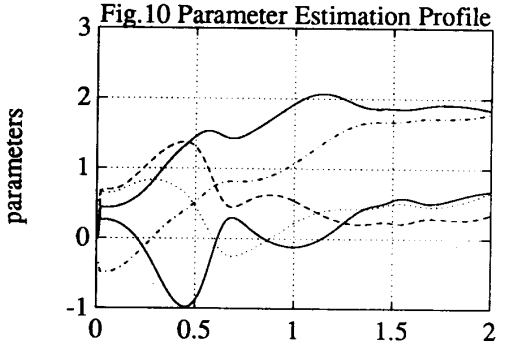
Fig.10 Parameter Estimation Profile
parameters
3
2
1
0
-1
0
0.5
1
1.5
2

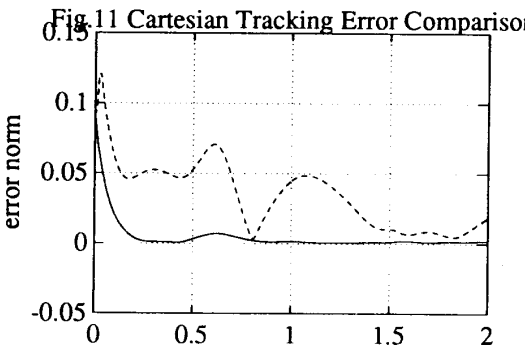
Fig.11 Cartesian Tracking Error Comparison
error norm
0.15
0.1
0.05
0
-0.05
0
0.5
1
1.5
2