Robotica (2001) volume 19, pp. 619–629. Printed in the United Kingdom © 2001 Cambridge University Press

Neuroadaptive control of elastic-joint robots using robust performance enhancement

C.J.B. Macnab and G.M.T. D'Eleuterio

University of Toronto Institute for Aerospace Studies, 4925 Dufferin St., Downsview, ON, M3H 5T6 (Canada)

(Received in Final Form: October 4, 1999)

SUMMARY

A neuroadaptive control scheme for elastic-joint robots is proposed that uses a relatively small neural network. Stability is achieved through standard Lyapunov techniques. For added performance, robust modifications are made to both the control law and the weight update law to compensate for only approximate learning of the dynamics. The estimate of the modeling error used in the robust terms is taken directly from the error of the network in modeling the dynamics at the currant state. The neural network used is the CMAC-RBF Associative Memory (CRAM), which is a modification of Albus's CMAC network and can be used for robots with elastic degrees of freedom. This results in a scheme that is computationally practical and results in good performance.

KEYWORDS: Neuroadaptive control; Elastic-joint robots; Neural network; Performance enhancement

1. INTRODUCTION

The precision control of elastic-joint robots has become of interest to a wide sector of the robot control community. Robots with harmonic drives in their joints are becoming increasingly popular because of their ability to provide in-line gear reduction at high efficiency. The price paid is structural flexibility in the joints. Flexibility results in both tracking error and unwanted vibrations when a simple controller, such as a PID controller, is used. An advanced control strategy that can effectively control the elastic degrees of freedom is highly desirable. One such control strategy is based on Lyapunov stability theory. It takes advantage of the mathematical form of highly geared elastic-joint robots in a method reffered to commonly as backstepping.[1] This method can be model-based,[2] robust,[3] adaptive,[4] or neuroadaptive.[5] We are specifically interested in achieving high tracking accuracy with an unknown model, limiting us to an adaptive, neuroadaptive, or robust approach. Unfortunately, an adaptive control approach carries excessive computational burden for the control of multilink manipulators. The neuroadaptive method potentially offers a practical approach; it relies on many simple calculations that can be executed in parallel. Even so, the size of the neural network must be kept within reason when serial processors are used. In addition, many neural-network schemes to date require very large training times, a procedure that may not be realistic for a robot on the job site. Robust methods offer an alternative solution; the control can be made robust to model uncertainty. However, a conservative estimate of the uncertainties must be used, which limits performance.

The aim of this paper is to develop a control method that delivers good performance using a small-sized neural network capable of learning only a rough approximation of the dynamics. In this way learning can take place quickly, and on-line computations at a high frequency become realistic. The modeling error of the neural network is measured on-line and used to make the performance robust to the shortcomings of the neural network. The method is still applicable, however, to larger neural networks. As more basis functions are added to the network, they will quite naturally contribute more to the control.

The solution to this problem is provided by developing a training/control method in conjunction with a neural-network structure. For training, a neuroadaptive backstepping scheme is used to guarantee stability, which uses a previously developed robust weight update known as "leakage".[6,7] The stabilizing weight update is augmented with an on-line learning term which speeds up the convergence of the system. Also, the error in the on-line learning is used as an estimate of model error, and is used in a robust control term in order to compensate for the neural network's shortcomings. In addition, the robust leakage term is modified to provide better learning. Thus, only enough basis functions are needed in the neural network to get a rough estimate of the dynamics and the robust terms compensate for the difference. This results in dramatically less basis functions neede in the neural network.

The Cerebellar Model Arithmetic Computer (CMAC) developed by Albus[8,9] is a type of neural network particularly suited to robotic control. It learns much faster than a multilayer perceptron and uses far less computational time than radial-basis-function associative memories. It has been successfully applied to rigid-robot control by many, including Miller[10,11] and Graham and D'Eleuterio.[12] In fact, it is Miller's learning scheme that is now commonly used. This control scheme is normally characterized by learning a trajectory quickly while using relatively few on-line basis functions. It would be desirable to recapture these controller qualities in the case of elastic-joint robots. Unfortunately, the CMAC scheme may result in instability when applied to robots with elastic degrees of freedom. To obtain stability, the CMAC is combined with radial basis functions (RBFs) to obtain the CMAC-RBF Associative Memory (CRAM).[13]

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

The result is CMAC-like structure capable of trajectory learning.

2. RIGID ROBOT

2.1. Control Lyapunov function

Consider the equations of motion for an n-link rigid robot:

$$\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}+\mathbf{f}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})=\mathbf{u}, \tag{1}$$

where

$\boldsymbol{\theta}\in\mathbb{R}^n$	contains the angles between links,
$\mathbf{M}(\boldsymbol{\theta})\in\mathbb{R}^{n\times n}$	is the inertia matrix,
$\mathbf{f}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})\in\mathbb{R}^n$	contains (nonlinear) forces,
$\mathbf{u}\in\mathbb{R}^n$	contains the control torques.

The state space representation of (1) can be given as

$$\mathbf{x}_1\triangleq\boldsymbol{\theta},\quad X_2\triangleq\dot{\boldsymbol{\theta}}. \tag{2}$$

Given the time-dependent desired trajectory and its derivatives in joint coordinates as $\boldsymbol{\theta}_d$, $\dot{\boldsymbol{\theta}}_d$, $\ddot{\boldsymbol{\theta}}_d$, then the auxiliary error, as in Slotine and Li,[14] is defined as

$$\mathbf{z}\triangleq\boldsymbol{\Lambda}(\mathbf{x}_1-\boldsymbol{\theta}_d)+(\mathbf{x}_2-\dot{\boldsymbol{\theta}}_d)=\boldsymbol{\Lambda}\mathbf{e}_1+\mathbf{e}_2,$$

where $\boldsymbol{\Lambda}$ is a positive-definite matrix. We are interested in driving the errors in $\mathbf{z}$ to zero using a neural-network controller. Consider the control Lyapunov function

$$V=\frac{1}{2}\mathbf{z}^T\mathbf{M}\mathbf{z}+\frac{1}{2\beta}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}. \tag{3}$$

where β is a positive constant, and $\tilde{\mathbf{w}}$ is a vector containing the weight errors of the neural network. Then the time derivative is

$$\dot{V}=\mathbf{z}^T\left(\mathbf{M}\dot{\mathbf{z}}+\frac{1}{2}\dot{\mathbf{M}}\mathbf{z}\right)-\frac{1}{\beta}\tilde{\mathbf{w}}^T\dot{\tilde{\mathbf{w}}}. \tag{4}$$

Consider that the unknown function in (4) can be written as

$$\mathbf{M}\dot{\mathbf{z}}+\frac{1}{2}\dot{\mathbf{M}}\mathbf{z}=\mathbf{M}(\boldsymbol{\Lambda}\mathbf{e}_2-\ddot{\boldsymbol{\theta}}_d)-\mathbf{f}+\mathbf{u}+\frac{1}{2}\dot{\mathbf{M}}\mathbf{z}. \tag{5}$$

We are interested in using a neural network to model the unknown (nonlinear) functions in (5). The output of an associative-memory neural network with n inputs and m basis functions can be described by $\boldsymbol{\Gamma}\hat{\mathbf{w}}$. The term $\Gamma\in\mathcal{R}^{n\times m}$ contains the state dependant basis functions. The vector $\hat{\mathbf{w}}\in\mathcal{R}^m$ contains the weights in the network, considered to be estimates of the "ideal" weights. The vector $\mathbf{w}$ is used to describe the "ideal" weights given a certain $\boldsymbol{\Gamma}$, that results in the best approximation. The error in the weights is then denoted $\tilde{\mathbf{w}}=\mathbf{w}-\hat{\mathbf{w}}$. Now (5) is rewritten with the neural model as

$$\mathbf{M}\dot{\mathbf{z}}+\frac{1}{2}\dot{\mathbf{M}}\mathbf{z}=\boldsymbol{\Gamma}(\mathbf{x}_1,\mathbf{x}_2,\boldsymbol{\theta}_d,\dot{\boldsymbol{\theta}}_d\,\ddot{\boldsymbol{\theta}}_d)\mathbf{w}+\mathbf{d}+\mathbf{u}, \tag{6}$$

where $\boldsymbol{\Gamma}\mathbf{w}$ is the best neural model (given $\boldsymbol{\Gamma}$) and $\mathbf{d}$ is the disturbance error due to modeling limitations of a finite number of basis functions in $\boldsymbol{\Gamma}$. Neural-networks are capable of uniform approximation of nonlinear functions within a certain region of input space. Within this region of uniform approximation $\mathbf{d}$ will be bounded so that $\mathbf{d}\leqslant d_{max}$. To use adaptive control techniques (4) is rewritten using the relation $\mathbf{w}=\tilde{\mathbf{w}}+\tilde{\mathbf{w}}$ as follows:

$$\dot{V}=\mathbf{z}^T(\boldsymbol{\Gamma}\tilde{\mathbf{w}}+\mathbf{d}+\mathbf{u})+\tilde{\mathbf{w}}^T\left(\boldsymbol{\Gamma}^T\mathbf{z}-\frac{1}{\beta}\dot{\hat{\mathbf{w}}}\right). \tag{7}$$

The task is now to choose a control $\mathbf{u}$ and weight update $\dot{\hat{\mathbf{w}}}$ to ensure $\dot{V}<0$, which ensures asymptotic stability. Also of great interest are modifications to $\mathbf{u}$ and $\dot{\hat{\mathbf{w}}}$ that improve the performance without effecting the stability.

2.2. Stability

2.2.1. Dead-band. Consider the control

$$\mathbf{u}=-\boldsymbol{\Gamma}\tilde{\mathbf{w}}-\mathbf{G}\mathbf{z}, \tag{8}$$

where $\mathbf{G}$ is positive definite, with weight update

$$\dot{\hat{\mathbf{w}}}=\beta\Gamma^T\mathbf{z}, \tag{9}$$

then if $\mathbf{d}=\mathbf{0}$, *i.e.* perfect modeling capability is assumed as in adaptive control,

$$\dot{V}=-\mathbf{z}^T\mathbf{G}\mathbf{z}. \tag{10}$$

Thus $\dot{V}\leqslant 0$, and is only negative semidefinite because there is no dependence on $\tilde{\mathbf{w}}$. According to a standard Lyapunov-like analysis,[15] $\mathbf{z}\rightarrow 0$ and $\dot{\hat{\mathbf{w}}}\rightarrow 0$ as $t\rightarrow\infty$. In other words, the error will go to zero, but the model will not necessarily converge to the correct weights. Note that $\mathbf{d}$ will always exist in neural-network control because, even in simulation, a finite number of basis functions will have approximation errors. In this case

$$\dot{V}=-\mathbf{z}^T\mathbf{G}\mathbf{z}+\mathbf{z}^T\mathbf{d}. \tag{11}$$

If we know the disturbance term is bounded, *i.e.* we are in the region of uniform approximation, such that

$$\|\mathbf{d}\|\leqslant d_{max}, \tag{11}$$

then

$$\dot{V}\leqslant\|\mathbf{z}\|(\|\mathbf{z}\|\lambda_{\min}(\mathbf{G})-d_{\max}), \tag{12}$$

where $\lambda_{\min}(\mathbf{G})$ is the minimum eigenvalue of $\mathbf{G}$ (assuming $\mathbf{G}$ is chosen to be symmetric). When $\|\mathbf{z}\|<d_{max}/\lambda_{\min}(\mathbf{G})$ then $\dot{V}$ is no longer less than zero. In this case we get what is referred to as *parameter drift*. That is, when the error gets close to zero, the parameters drift towards infinity. To prevent this effect, we can simply implement a dead band in the weight update

$$\dot{\hat{\mathbf{w}}}=\begin{cases}\beta\boldsymbol{\Gamma}^T\mathbf{z} & \text{if } \|\mathbf{z}\|>d_{\max}/\lambda_{\min}(\mathbf{G})\\ 0 & \text{otherwise.}\end{cases} \tag{13}$$

Such a method has been used for some time in adaptive control. One drawback, however, is that normally a very conservative estimate of d_{max} must be used.

2.2.2. Robust control. Consider the modified robust control

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

$$\mathbf{u} = -\Gamma\tilde{\mathbf{w}} - \frac{d_{\max}\mathbf{z}}{\|\mathbf{z}\|} - \mathbf{G}\mathbf{z}, \tag{14}$$

where the second term $d_{\max}\mathbf{z}/\|\mathbf{z}\|$ is referred to as *min-max* control,[16] and we have the same weight update as before,

$$\dot{\hat{\mathbf{w}}} = \beta\Gamma^T\mathbf{z}. \tag{15}$$

Then

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}^T\mathbf{d} - \frac{d_{\max}\mathbf{z}^T\mathbf{z}}{\|\mathbf{z}\|}, \tag{16}$$

and

$$\dot{V} \leq -\|\mathbf{z}\|(\lambda_{\min}(\mathbf{G})\|\mathbf{z}\| + d_{\max} - \|\mathbf{d}\|) \tag{17}$$

Since $d_{\max} \geq \|\mathbf{d}\|$ we know $\dot{V} \leq 0$. In reality it is undesirable (and perhaps impossible) to have such a sharp transition in the control force since it results in torque *chatter* when the error is small. That is why a robust term like

$$\mathbf{u} = -\Gamma\hat{\mathbf{w}} - \frac{d_{max}\mathbf{z}}{\|\mathbf{z}\| + \varepsilon} - \mathbf{G}\mathbf{z} \tag{18}$$

is used where ε is a small positive number. This is referred to as *saturation-type* control.[17] Now

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}^T\mathbf{d} - \mathbf{z}^T\mathbf{z}\frac{d_{max}}{\|\mathbf{z}\| + \varepsilon}, \tag{19}$$

$$\leq -\|\mathbf{z}\|^2\lambda_{\min}(\mathbf{G}) + \|\mathbf{z}\|d_{\max} - \frac{\|\mathbf{z}\|^2 d_{max}}{\|\mathbf{z}\| + \varepsilon}, \tag{20}$$

$$\leq -\|\mathbf{z}\|^2\lambda_{\min}(\mathbf{G}) + \frac{\|\mathbf{z}\|\varepsilon}{\|\mathbf{z}\| + \varepsilon} d_{max}. \tag{21}$$

The point is that when $\varepsilon \ll 1$ then

$$\frac{\|\mathbf{z}\|\varepsilon}{\|\mathbf{z}\| + \varepsilon} \ll 1,$$

so that the new disturbance term is made much smaller than the original **d**. However, when

$$\|\mathbf{z}\| \leq \sqrt{\frac{\varepsilon^2/2 + \varepsilon d_{\max}}{\lambda_{\min}(\mathbf{G})}} - \frac{\varepsilon}{2}, \tag{22}$$

we no longer have a negative-semidefinite Lyapunov derivative, and parameter drift will again occur. To eliminate this, one can choose a dead band based on (22), which is much smaller than before. To obtain even smoother transition in the torque, the robust control term can be choosen to be

$$\mathbf{u} = -\Gamma\tilde{\mathbf{w}} - \frac{d_{\max}^2\mathbf{z}}{d_{\max}\|\mathbf{z}\| + \varepsilon} - \mathbf{G}\mathbf{z}. \tag{23}$$

2.2.3. Robust weight update. Consider again the original control

$$\mathbf{u} = -\Gamma\tilde{\mathbf{w}} - \mathbf{G}\mathbf{z}, \tag{24}$$

and modified robust weight update

$$\dot{\hat{\mathbf{w}}} = \beta(\Gamma^T\mathbf{z} - \nu\|\mathbf{z}\|\tilde{\mathbf{w}}). \tag{25}$$

Adding a penalty of form $-\tilde{\mathbf{w}}$ to adaptive update rules was the idea of Ioannou,[6] and was referred to as *leakage*, since it creates a "leaky" integrator. Narenda and Annaswamy[7] multiplied this by $\|\mathbf{z}\|$, a technique often referred to as ε-*mod*. As we shall see, this eliminates the parameter drift effect. The resulting Lyapunov derivative is

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}^T\mathbf{d} + \nu\|\mathbf{z}\|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}. \tag{26}$$

Using the fact that $\tilde{\mathbf{w}} = \mathbf{w} - \hat{\mathbf{w}}$,

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}^T\mathbf{d} + \nu\|\mathbf{z}\|\tilde{\mathbf{w}}^T\mathbf{w} - \nu\|\mathbf{z}\|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}. \tag{27}$$

We assume disturbance and ideal weights are bounded as

$$\|\mathbf{d}\| \leq d_{max}, \qquad \|\mathbf{w}\| \leq \omega_{max},$$

then using the relations

$$\tilde{\mathbf{w}}^T\mathbf{w} \leq \|\tilde{\mathbf{w}}\|\|\mathbf{w}\|, \qquad \tilde{\mathbf{w}}^T\tilde{\mathbf{w}} = \|\tilde{\mathbf{w}}\|^2,$$

we have

$$\dot{V} \leq -\|\mathbf{z}\|[\lambda_{\min}(\mathbf{G})\|z\| - d_{\max} + \nu\|\tilde{\mathbf{w}}\|(\|\tilde{\mathbf{w}}\| - \omega_{\max})], \tag{28}$$

and

$$\dot{V} \leq -\|\mathbf{z}\|\left[\lambda_{\min}(\mathbf{G})\|\mathbf{z}\| - d_{\max} + \nu\left(\|\tilde{\mathbf{w}}\| - \frac{\omega_{\max}}{2}\right)^2 - \frac{\nu\omega_{\max}^2}{4}\right]. \tag{29}$$

We can ensure that $\dot{V} < 0$ when

$$\|\mathbf{z}\| > (\nu\omega_{\max}^2/4 + d_{\max})/\lambda_{\min}(\mathbf{G}), \tag{30}$$

or

$$\|\tilde{\mathbf{w}}\| > \omega_{\max}/2 + \sqrt{\omega_{\max}^2/4 + d_{\max}/\nu}. \tag{31}$$

Thus outside a region $\mathcal{B}$ defined by (30) and (31), $\dot{V} < 0$. As $t \to \infty$ then $\|\mathbf{z}\|, \|\tilde{\mathbf{w}}\| \to \mathcal{B}$. This also implies that V is bounded, in turn implying that $\|\mathbf{z}\|\|\tilde{\mathbf{w}}\|$ have *uniform ultimate boundedness* (UUB).[18] In this way we have assured that the weight remains bounded inside a region, and have thus eliminated the effect of parameter drift. Notice that the size of $\|\mathbf{z}\|$ as $t \to \infty$ can be made arbitrarily small by picking large enough **G**. However, no similar method exists for $\|\tilde{\mathbf{w}}\|$. Thus the neural network will still not learn the correct weights, although this is not the control objective. To learn the correct weights, a persistence of excitation condition on the input is needed.[15]

2.3. Performance

Lyapunov functions are excellent for stability analysis. Quantifying performance is more difficult. In general, the larger the magnitude of $\dot{V} < 0$, the better the performance. In this paper, performance-oriented modifications will be made that do not affect the stability analysis, and performance improvement will simply be demonstrated using simulations.

2.3.1. Optimal control. Consider the case where $\mathbf{d} = \mathbf{0}$, and apply control

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

$$\mathbf{u}=\begin{cases}-\mathbf{\Gamma}\hat{\mathbf{w}}-\mathbf{G}\mathbf{z} & \text{if } \mathbf{z}^T\mathbf{\Gamma}\hat{\mathbf{w}}\geqslant 0\\ -\mathbf{G}\mathbf{z} & \text{otherwise.}\end{cases} \tag{32}$$

Since this guarantees that $\dot{V}\leqslant 0$, no change in the stability analysis has occurred. However, the effect has been to not apply the control when $\dot{V}$ is less than zero already. The term *optimal* can be used since it limits the control effort.[19] Also, the performance is improved, as can be seen by demonstration in Section 5.

Again, sharp changes in the control are usually impossible, or at least undesirable. For a modification to keep the torques smooth consider the function

$$s=\begin{cases}1 & \text{if } \mathbf{z}^T\mathbf{\Gamma}\tilde{\mathbf{w}}>0,\\ \frac{1}{\gamma}\mathbf{z}^T\mathbf{\Gamma}\tilde{\mathbf{w}}+1 & \text{if } -\gamma>\mathbf{z}^T\mathbf{\Gamma}\tilde{\mathbf{w}}\leqslant 0,\\ 0 & \text{otherwise,}\end{cases} \tag{33}$$

where γ is a positive constant. The smooth control

$$\mathbf{u}=-s\mathbf{\Gamma}\hat{\mathbf{w}}-\mathbf{G}\mathbf{z} \tag{34}$$

replaces (32).

2.3.2. On-line, direct learning. In previous methods of neural network control, like Miller's CMAC training scheme,[10,11] the neural network learns the inverse dynamics of the current state directly and relies on generalization for learning of the desired trajectory's inverse dynamics. This type of learning was modified for continuous time modeling and shown to be Lyapunov stable by Macnab and D'Eleuterio.[13] A similar type of scheme was developed for adaptive control by Slotine and Li,[20] which uses a filter to eliminate acceleration measurement as well as a learning gain matrix calculated using a recursive least squares estimate.

Assuming perfect modeling ability, we have

$$\mathbf{M}\dot{\mathbf{z}}+\frac{1}{2}\dot{\mathbf{M}}\mathbf{z}=\mathbf{\Gamma}(\mathbf{x}_1,\,\mathbf{x}_2,\,\boldsymbol{\theta}_d,\,\dot{\boldsymbol{\theta}}_d,\,\ddot{\boldsymbol{\theta}}_d)\mathbf{w}+\mathbf{u}. \tag{35}$$

On the desired trajectory, $\mathbf{e}_1=\mathbf{e}_2=\mathbf{z}_1=\mathbf{0}$ and (35) becomes

$$\mathbf{M}(\dot{\mathbf{x}}_2-\ddot{\boldsymbol{\theta}}_d)=\mathbf{\Gamma}\mathbf{w}+\mathbf{u}. \tag{36}$$

To ensure $(\dot{\mathbf{x}}_2-\ddot{\boldsymbol{\theta}}_d)=0$, *i.e.* to stay on the trajectory, the proper control is $\mathbf{u}=-\mathbf{\Gamma}\mathbf{w}$. Thus at $\mathbf{z}_1=\mathbf{0}$, we have $-\mathbf{\Gamma}\mathbf{w}$ as the *inverse dynamics*, that is the control force needed to achieve a desired acceleration. The trick is to assume we are already following the desired trajectory, and by denoting the basis functions at this state as $\mathbf{\Gamma}'$ we have

$$\mathbf{\Gamma}'(\mathbf{x}_1,\,\mathbf{x}_2,\,\mathbf{x}_1,\,\mathbf{x}_2,\,\dot{\mathbf{x}}_2)\mathbf{w}=-\mathbf{u}. \tag{37}$$

The weights are easy to learn using gradient descent:

$$\dot{\hat{\mathbf{w}}}=\eta\mathbf{\Gamma}'^T(-\mathbf{u}-\mathbf{\Gamma}'\tilde{\mathbf{w}}). \tag{38}$$

The idea is that learning done at $\mathbf{\Gamma}'$ will genererlize to $\mathbf{\Gamma}$ on the actual desired trajectory. Note that $\dot{\mathbf{x}}_2$ is an acceleration. If acceleration cannot be measured, it must be estimated. Yet, this estimate is not being used to provide stability, but rather performance. Thus a noisy estimate is not necessarily a problem.

Let us examine the effect on stability, assuming we have control

$$\mathbf{u}=-\mathbf{\Gamma}\tilde{\mathbf{w}}-\mathbf{G}\mathbf{z}, \tag{39}$$

and modified weight update that includes on-line, direct learning

$$\dot{\hat{\mathbf{w}}}=\beta[\mathbf{\Gamma}^T\mathbf{z}-\nu\|\mathbf{z}\|\hat{\mathbf{w}}+\eta\|\mathbf{z}\|\mathbf{\Gamma}'^T(-\mathbf{u}-\mathbf{\Gamma}'\tilde{\mathbf{w}})]. \tag{40}$$

Then using $\tilde{\mathbf{w}}=\mathbf{w}-\hat{\mathbf{w}}$,

$$\begin{aligned}\dot{V}=&-\mathbf{z}^T\mathbf{G}\mathbf{z}+\mathbf{z}^T\mathbf{d}+\|\mathbf{z}\|\tilde{\mathbf{w}}^T[\nu\mathbf{w}+\eta\mathbf{\Gamma}'^T(-\mathbf{u}-\mathbf{\Gamma}'\mathbf{w})]\\ &-\|\mathbf{z}\|\tilde{\mathbf{w}}^T(\mathbf{I}\nu+\eta\mathbf{\Gamma}'^T\mathbf{\Gamma}')\tilde{\mathbf{w}}.\end{aligned} \tag{41}$$

We can denote

$$\boldsymbol{\tau}\triangleq-\mathbf{u}-\mathbf{\Gamma}'\mathbf{w}, \tag{42}$$

where $\boldsymbol{\tau}$ is a term like $\mathbf{d}$ that represents shortcomings in the basis functions, and as well as, in this case, any errors from estimation of $\dot{\mathbf{x}}_2$. Since $\mathbf{\Gamma}'^T\mathbf{\Gamma}'$ is a positive-semidefinite matrix, it can help the performance, but the stability based on a worst-case scenario is derived from

$$\begin{aligned}\dot{V}\leqslant&-\|\mathbf{z}\|[\lambda_{\min}(\mathbf{G})\|\mathbf{z}\|-d_{\max}+\|\tilde{\mathbf{w}}\|\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau})\\ &+\nu\|\tilde{\mathbf{w}}\|(\|\tilde{\mathbf{w}}\|-\omega_{\max})],\end{aligned} \tag{43}$$

resulting in slightly larger maximum bounds than in (30) and (31).

2.3.3. Modeling error robust compensation. We will now take advantage of the error produced in the Miller style learning to create a robust control term that compenstaes for the imperfect modeling ability of the neural network. This is the first of the original contributions in this paper. The scheme is illustrated for clarity in Figure 1. Let us define the error

$$\mathbf{r}\triangleq-\mathbf{u}-\mathbf{\Gamma}'\tilde{\mathbf{w}}. \tag{44}$$

In the on-line, direct learning the control $\mathbf{u}$ is used as a measurement of $\mathbf{\Gamma}'\mathbf{w}$ to learn this function directly with gradient descent. Thus when the function has been learned, *i.e.* $\mathbf{\Gamma}'\hat{\mathbf{w}}$ approximates $\mathbf{\Gamma}'\mathbf{w}$, we can write

$$\mathbf{r}\approx-\mathbf{u}-\mathbf{\Gamma}'\mathbf{w}\approx\mathbf{d}. \tag{45}$$

The idea, then, is to use $\mathbf{r}$ as an estimate of how well the neural network is capable of learning the functions along the desired trajectory. Thus a robust term can be used in the control which leads to

$$\mathbf{u}=-\mathbf{\Gamma}\hat{\mathbf{w}}-\frac{\|\mathbf{r}\|\mathbf{z}}{\|\mathbf{z}\|+\varepsilon}-\mathbf{G}\mathbf{z}, \tag{46}$$

where $\|\mathbf{r}\|$ must obviously be calculated using a previous value or values of the control $\mathbf{u}$. In this paper, a simple time delay is assumed of the same discreate time step, h, at which neural network is operating, so that $\|\mathbf{r}\|$ is in reality $\|\mathbf{r}(t-h)\|$. Early in learning when $\mathbf{\Gamma}'\hat{\mathbf{w}}$ does not approximate $\mathbf{\Gamma}'\mathbf{w}$ yet.

$$\|\mathbf{r}\|>\|\mathbf{d}\|, \tag{47}$$

and later when $\mathbf{\Gamma}'\hat{\mathbf{w}}$ approximates $\mathbf{\Gamma}'\mathbf{w}$,

$$\|\mathbf{r}\|\approx\|\mathbf{d}\|. \tag{48}$$

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

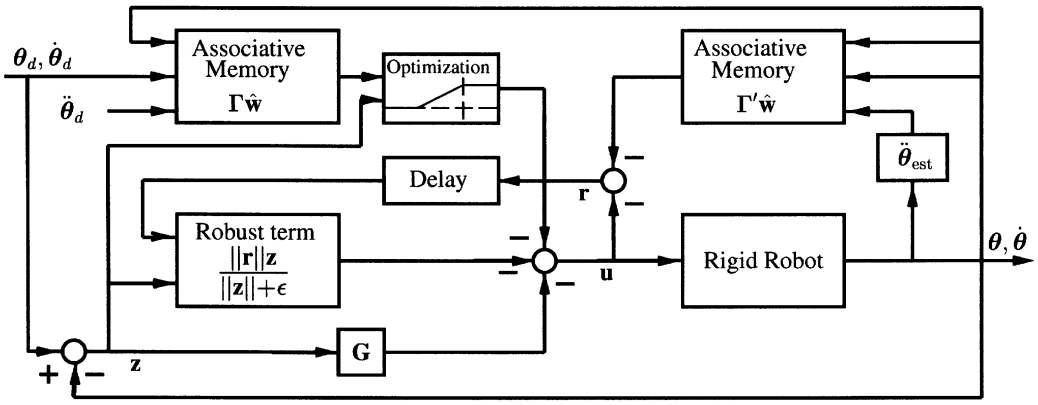

Fig. 1. High performance neurocontrol scheme for rigid robot.

So early in learning this acts simple as a robust control term that compensates for the fact that the dynamics are as yet unlearned. Later in learning, it compensates for the shortcomings of the approximation ability of the network.

Let us denote

$$\delta \triangleq d_{\max} - \frac{\|\mathbf{r}\|\|\mathbf{z}\|}{\|\mathbf{z}\|+\varepsilon}. \tag{49}$$

We can safely assume that later in learning

$$\delta_{\max} \ll d_{\max}, \tag{50}$$

so that the disturbance term in the steady state has been dramatically reduced. Combined with the previous modified weight updates, (40), the Lyapunov derivitive, continuing from (43), is now

$$\dot{V} \leqslant -\|\mathbf{z}\|\left[\lambda_{\min}(\mathbf{G})\|\mathbf{z}\| - \delta_{\max} + \left(\sqrt{v}\|\tilde{\mathbf{w}}\| - \frac{\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + v\omega_{\max}}{2\sqrt{v}}\right)^2 - \frac{(\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + v\omega_{\max})^2}{4v}\right].$$

We can ensure that $\dot{V} < 0$ when

$$\|\mathbf{z}\| > \frac{1}{\lambda_{\min}(\mathbf{G})}\left[\frac{(\eta\lambda_{\min}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + v\omega_{\max})^2}{4v} + \delta_{\max}\right], \tag{51}$$

or

$$\|\tilde{\mathbf{w}}\| > \frac{1}{\sqrt{v}}\left[\frac{(\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + v\omega_{\max}}{2\sqrt{v}} + \left(\delta_{\max} + \frac{(\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + v\omega_{\max})^2}{4v}\right)^{1/2}\right] \tag{52}$$

The result is that arbitrary performance criteria can be met with less cost in torque chatter than with a strictly robust type of scheme.

2.4. Modified robust weight update

This section introduces the second original contribution of the paper. The leakage term used to provide robustness in the weight update is not ideal. Qualitatively, it is trying to force the weights in $\hat{\mathbf{w}}$ to zero. Consider, instead

$$\dot{\hat{\mathbf{w}}} = \beta[\mathbf{\Gamma}^T\mathbf{z} - v\|\mathbf{z}\|(\hat{\mathbf{w}} - \boldsymbol{\alpha})]. \tag{53}$$

where $\boldsymbol{\alpha}$ is to be selected. If the robust control, (46), is used then the bounds for $\dot{V} < 0$ in equations (30), (31) become

$$\|\mathbf{z}\| > (v\|\mathbf{w} - \boldsymbol{\alpha}\|_{\max}^2/4 + \delta_{\max})/\lambda_{\min}(\mathbf{G}), \tag{54}$$

or

$$\|\tilde{\mathbf{w}}\| > \|\mathbf{w} - \boldsymbol{\alpha}\|_{\max}/2 + \sqrt{\|\mathbf{w} - \boldsymbol{\alpha}\|_{\max}^2/4 + \delta_{\max}/v}. \tag{55}$$

In the previous result, $\tilde{\mathbf{w}}$ could not be arbitrarily reduced. Now, however, by choosing $\boldsymbol{\alpha}$ close to $\hat{\mathbf{w}}$, the bound of $\tilde{\mathbf{w}}$ can be substancially reduced. It appears that $\boldsymbol{\alpha}$ must be a constant to guarantee stability. This will be useful when some type of off-line pretraining is used (a common technique) and then such an estimate $\boldsymbol{\alpha} = \hat{\mathbf{w}}_0$ exists at the start of the on-line training phase. Consider, however, an on-line modification of the form

$$\boldsymbol{\alpha} = \hat{\mathbf{w}}, \qquad \text{when } \|\mathbf{z}\| > \|\mathbf{r}\|/\lambda_{\min}(\mathbf{G}), \tag{56}$$

and

$$\dot{\boldsymbol{\alpha}} = \mathbf{0}, \qquad \text{otherwise.} \tag{57}$$

Thus, when $\|\mathbf{z}\| > \|\mathbf{r}\|/\lambda_{\min}(\mathbf{G})$, no leakage at all is applied. In this case, stability is guaranteed for $\|\mathbf{z}\| > \delta_{\max}/\lambda_{\min}(\mathbf{G})$. We argue that $\|\mathbf{r}\|$ is a very conservative estimate for $\delta_{\max}$, and thus stability is guaranteed. When $\|\mathbf{z}\| \leq \|\mathbf{r}\|/\lambda_{\min}(\mathbf{G})$, $\boldsymbol{\alpha}$ is a constant that is somewhat close to $\hat{\mathbf{w}}$ and the bound for $\|\tilde{\mathbf{w}}\|$, is reduced.

3. ELASTIC-JOINT ROBOT

The advantage in developing such a Lyapunov based control strategy for a rigid robot is that it becomes straightforward to apply it to an elastic-joint robot using the technique of backstepping.[1] The equations for an n-link elastic-joint robot, shown in Figure 2, with large gear ratios can be written as in Spong[21] and they are

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

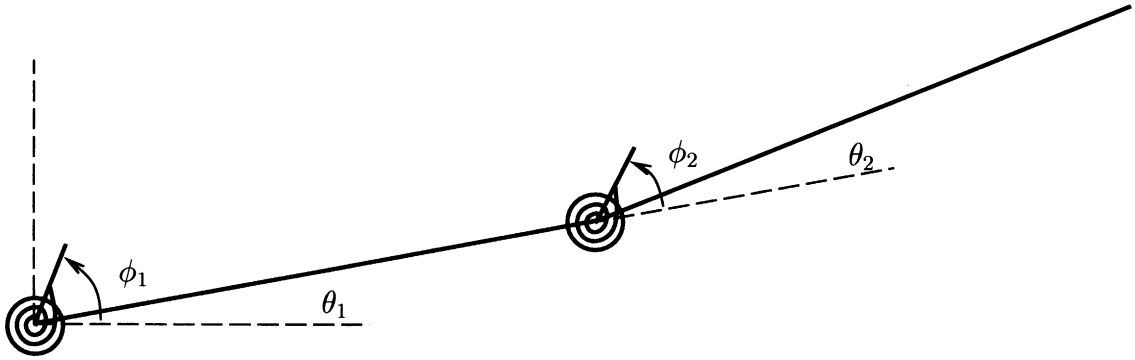

Fig. 2. Planar, two-link elastic-joint robot.

$$\mathbf{M}\ddot{\boldsymbol{\theta}}+\mathbf{f}_1(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})+\mathbf{K}(\boldsymbol{\theta}-\boldsymbol{\phi})=\mathbf{0}, \tag{58}$$

$$\mathbf{J}\ddot{\boldsymbol{\phi}}+\mathbf{f}_2(\boldsymbol{\phi},\dot{\boldsymbol{\phi}})-\mathbf{K}(\boldsymbol{\theta}-\boldsymbol{\phi})=\mathbf{u}, \tag{59}$$

where new variables introduced are $\boldsymbol{\phi}$ which contain the motor angles after gear reduction, $\mathbf{f}_1$, $\mathbf{f}_2 \in \mathbb{R}^n$ which contain (nonlinear) forces, $\mathbf{K} \in \mathbb{R}^{n\times n}$, a constant diagonal matrix of joint stiffnesses, and $\mathbf{J} \in \mathbb{R}^{n\times n}$, a constant diagonal matrix of joint inertias after gear reduction. Defining the state variables as

$$\mathbf{x}_1 \triangleq \boldsymbol{\theta},\ \mathbf{x}_2 \triangleq \dot{\boldsymbol{\theta}},\ \mathbf{x}_3 \triangleq \boldsymbol{\phi},\ \mathbf{x}_4 \triangleq \dot{\boldsymbol{\phi}},$$

then the errors are defined as

$$\mathbf{e}_1 \triangleq \mathbf{x}_1 - \boldsymbol{\theta}_d \qquad \mathbf{e}_2 \triangleq \mathbf{x}_2 - \dot{\boldsymbol{\theta}}_d,$$

and

$$\mathbf{z}_1 \triangleq \boldsymbol{\Lambda}\mathbf{e}_1+\mathbf{e}_2, \qquad \mathbf{z}_2 \triangleq \mathbf{x}_3 - \mathbf{x}_3^*, \qquad \mathbf{z}_3 \triangleq \mathbf{x}_4 - \mathbf{x}_4^*,$$

where $\mathbf{x}_3^*$, $\mathbf{x}_4^*$ are considered to be virtual control terms and must be calculated on-line.

The unknown functions can be written as

$$\mathbf{K}^{-1}\mathbf{M}\dot{\mathbf{z}}_1+\frac{1}{2}\mathbf{K}^{-1}\dot{\mathbf{M}}\mathbf{z}_1=\boldsymbol{\Gamma}_1\mathbf{w}_1-\mathbf{x}_1+(\mathbf{z}_2+\mathbf{x}_3^*), \tag{60}$$

$$\dot{\mathbf{z}}_2=\boldsymbol{\Gamma}_2\mathbf{w}_2-\mathbf{x}_2+(\mathbf{z}_3+\mathbf{x}_4^*), \tag{61}$$

$$\mathbf{J}\dot{\mathbf{z}}_3=\boldsymbol{\Gamma}_3\mathbf{w}_3+\mathbf{u}, \tag{62}$$

where the terms $\boldsymbol{\Gamma}_i\mathbf{w}_i$ are used to represent

$$\boldsymbol{\Gamma}_1\mathbf{w}_1=\mathbf{K}^{-1}\mathbf{M}(\mathbf{e}_2-\ddot{\mathbf{q}}_d)-\mathbf{K}^{-1}\mathbf{f}_1+\frac{1}{2}\mathbf{K}^{-1}\dot{\mathbf{M}}\mathbf{z}_1, \tag{63}$$

$$\boldsymbol{\Gamma}_2\mathbf{w}_2=-\dot{\mathbf{x}}_3^*+\mathbf{x}_2, \tag{64}$$

$$\boldsymbol{\Gamma}_3\mathbf{w}_3=-\mathbf{J}\dot{\mathbf{x}}_4^*-\mathbf{f}_2+\mathbf{K}(\mathbf{x}_1-\mathbf{x}_3), \tag{65}$$

and the explicit dependencies are

$$\boldsymbol{\Gamma}_1(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d),$$

$$\boldsymbol{\Gamma}_2(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d, \boldsymbol{\theta}_d^{(3)}),$$

$$\boldsymbol{\Gamma}_3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d, \boldsymbol{\theta}_d^{(3)}, \boldsymbol{\theta}_d^{(4)}),$$

In this paper, the basis functions will not depend on $\boldsymbol{\theta}_d^{(3)}$, $\boldsymbol{\theta}_d^{(4)}$. This is justified in a neural-network scheme because the input variables simply act to index the memory. As long as the inputs $\boldsymbol{\theta}_d$, $\dot{\boldsymbol{\theta}}_d$, $\ddot{\boldsymbol{\theta}}_d$ uniquely define the trajectory, no other imformation is needed.

3.1. Stability

A controller can be defined as follows: the stabilizing virtual control terms are

$$\mathbf{x}_3^* = -\boldsymbol{\Gamma}_1\hat{\mathbf{w}}_1+\mathbf{x}_1-\mathbf{G}_1\mathbf{z}_1, \tag{66}$$

$$\mathbf{x}_4^* = -\boldsymbol{\Gamma}_2\hat{\mathbf{w}}_2+\mathbf{x}_2-\mathbf{z}_1\mathbf{G}_2\mathbf{z}_2, \tag{67}$$

where terms $\mathbf{G}_1$, $\mathbf{G}_2$ are positive-definite gain matrices. The stabilizingt control term is

$$\mathbf{u}=-\boldsymbol{\Gamma}_3\hat{\mathbf{w}}_3-\mathbf{z}_2-\mathbf{G}_3\mathbf{z}_3. \tag{68}$$

The stabilizing weight update laws are

$$\dot{\hat{\mathbf{w}}}_i=\beta(\boldsymbol{\Gamma}_i^T\mathbf{z}_i-\nu\|\mathbf{z}\|\hat{\mathbf{w}}_i), \qquad \text{for } i=1, 2, 3 \tag{69}$$

where $\mathbf{z}=\text{col}\{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3\}$, β is a positive constant used as an adaption or learning gain, and ν is a small positive constant. To see this, consider the Lyapunov function,

$$V=\frac{1}{2}\mathbf{z}_1^T\mathbf{K}^{-1}\mathbf{M}\mathbf{z}_1+\frac{1}{2}\mathbf{z}_2^T\mathbf{z}_2+\frac{1}{2}\mathbf{z}_3^T\mathbf{J}\mathbf{z}_3+\frac{1}{2}\sum_{i=1}^{3}\frac{1}{\beta}\tilde{\mathbf{w}}_i^T\tilde{\mathbf{w}}_i. \tag{70}$$

The time derivative is

$$\dot{V}=\mathbf{z}_1^T(\boldsymbol{\Gamma}_1\mathbf{w}_1+\mathbf{d}_1-\mathbf{x}_1+\mathbf{z}_2+\mathbf{x}_3^*)+\mathbf{z}_2^T(\boldsymbol{\Gamma}_2\mathbf{w}_2+\mathbf{d}_2-\mathbf{x}_2+\mathbf{z}_3+\mathbf{x}_4^*)$$

$$+\mathbf{z}_3^T(\boldsymbol{\Gamma}_3\mathbf{w}_3+\mathbf{d}_3+\mathbf{u})-\sum_{i=1}^{3}\frac{1}{\beta}\tilde{\mathbf{w}}_i^T\dot{\hat{\mathbf{w}}}_i, \tag{71}$$

where the terms $\mathbf{d}_i$ are disturbances due to modeling limitations in Γ_i. To use adaptive control the equation is rewritten using the relation $\mathbf{w}_i=\tilde{\mathbf{w}}_i+\hat{\mathbf{w}}_i$ as follows

$$\dot{V}=\mathbf{z}_2^T(\boldsymbol{\Gamma}_1\hat{\mathbf{w}}_1+\mathbf{d}_1-\mathbf{x}_1+\mathbf{z}_2+\mathbf{x}_3^*)+\mathbf{z}_2^T(\boldsymbol{\Gamma}_2\hat{\mathbf{w}}_2+\mathbf{d}_2-\mathbf{x}_2+\mathbf{z}_3+\mathbf{x}_4^*)$$

$$+\mathbf{z}_3^T(\boldsymbol{\Gamma}_3\hat{\mathbf{w}}_3+\mathbf{d}_3+\mathbf{u})+\sum_{i=1}^{3}\tilde{\mathbf{w}}_i^T(\boldsymbol{\Gamma}_i^T\mathbf{z}_i-\frac{1}{\beta}\dot{\hat{\mathbf{w}}}_i). \tag{72}$$

The control terms (66)–(68) and parameter updates (69) substituted in (72) result in

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}^T\mathbf{d} + \nu\|\mathbf{z}\|\tilde{\mathbf{w}}^T\hat{\mathbf{w}}, \tag{73}$$

where

$$\mathbf{G} = \text{diag}\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3\},$$
$$\mathbf{d} = \text{col}\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\},$$
$$\mathbf{w} = \text{col}\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}.$$

In the same manner as before, we can ensure that $\dot{V} < 0$ when

$$\|\mathbf{z}\| > (\nu\omega_{\max}^2/4 + d_{\max})/\lambda_{\min}(\mathbf{G}), \tag{74}$$

or

$$\|\tilde{\mathbf{w}}\| > \omega_{\max}/2 + \sqrt{\omega_{\max}^2/4 + d_{\max}/\nu}. \tag{75}$$

Thus $\dot{V}(\mathbf{z}, \tilde{\mathbf{w}}) < 0$ outside a compact set $\mathcal{B}$. The same conclusions apply as for the rigid case. In particular, we can ensure that the state converges arbitrarily close to the trajectory by increasing $\mathbf{G}$, but we cannot not guarantee that the neural network weights will come close to their correct values.

3.2. Performance

We can add our performance modifications quite simply. Consider with perfect modeling that

$$\mathbf{K}^{-1}\mathbf{M}\dot{\mathbf{z}}_1 + \frac{1}{2}\mathbf{K}^{-1}\dot{\mathbf{M}}\mathbf{z}_1 = \mathbf{\Gamma}_1(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d)\mathbf{w}_1 - \mathbf{x}_1 + \mathbf{x}_3. \tag{76}$$

When $\mathbf{e}_1 = \mathbf{e}_2 = \mathbf{z}_1 = \mathbf{z}_2 = \mathbf{0}$, the function is evaluated at the actual, instead of the desired, states, and for ease of presentation it will be denoted $\mathbf{\Gamma}'_1$. Then we have

$$\mathbf{\Gamma}'_1(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_2, \dot{\mathbf{x}}_2)\mathbf{w}_1 = \mathbf{x}_1 - \mathbf{x}_3, \tag{77}$$

and the error function

$$\mathbf{r}_1 = (\mathbf{x}_1 - \mathbf{x}_3) - \mathbf{\Gamma}'_1\hat{\mathbf{w}}_1. \tag{78}$$

The same method can be used for $\hat{\mathbf{w}}_3$ since ideally

$$\mathbf{J}\dot{\mathbf{z}}_3 = \mathbf{\Gamma}_3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d)\mathbf{w}_3 + \mathbf{u}. \tag{79}$$

Again, at zero error the result is

$$\mathbf{\Gamma}'_3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_1, \mathbf{x}_2, \dot{\mathbf{x}}_2)\mathbf{w}_3 = -\mathbf{u}, \tag{80}$$

with error function

$$\mathbf{r}_3 = -\mathbf{u} - \mathbf{\Gamma}'_3\hat{\mathbf{w}}_3. \tag{81}$$

The updates for $\hat{\mathbf{w}}_2$ can be done in a more straightforward manner since

$$\mathbf{\Gamma}_2\mathbf{w}_2 = -\dot{\mathbf{x}}_3^* + \mathbf{x}_2. \tag{82}$$

The value of $\dot{\mathbf{x}}_3^*$ can be estimated by differentiation. The error is

$$\mathbf{r}_2 = (-\dot{\mathbf{x}}_3^* + \mathbf{x}_2) - \mathbf{\Gamma}_2\hat{\mathbf{w}}_2. \tag{83}$$

The optimal control term factor is calculated as before,

$$s_i = \begin{cases} 1 & \text{if } \mathbf{z}_i^T\mathbf{\Gamma}_i\hat{\mathbf{w}}_i > 0, \\ \mathbf{z}_i^T\mathbf{\Gamma}_i\hat{\mathbf{w}}_i/\gamma_i + 1 & \text{if } -\gamma_i < \mathbf{z}_i^T\mathbf{\Gamma}_i\hat{\mathbf{w}}_i \leq 0, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } i = 1, 2, 3, \tag{84}$$

Now consider the virtual controls and the actual control

$$\mathbf{x}_3^* = -s_1\mathbf{\Gamma}_1\hat{\mathbf{w}}_1 - \frac{\|\mathbf{r}_1\|\mathbf{z}_1}{\|\mathbf{z}_1\| + \varepsilon} + \mathbf{x}_1 - \mathbf{G}_1\mathbf{z}_1, \tag{85}$$

$$\mathbf{x}_4^* = -s_2\mathbf{\Gamma}_2\hat{\mathbf{w}}_2 - \frac{\|\mathbf{r}_2\|\mathbf{z}_2}{\|\mathbf{z}_2\| + \varepsilon} + \mathbf{x}_2 - \mathbf{z}_1 - \mathbf{G}_2\mathbf{z}_2, \tag{86}$$

$$\mathbf{u} = -s_3\mathbf{\Gamma}_3\hat{\mathbf{w}}_3 - \frac{\|\mathbf{r}_3\|\mathbf{z}_3}{\|\mathbf{z}_3\| + \varepsilon} - \mathbf{z}_2 - \mathbf{G}_3\mathbf{z}_3, \tag{87}$$

along with the weight updates,

$$\dot{\hat{\mathbf{w}}} = \beta(\mathbf{\Gamma}_i^T\mathbf{z}_i - \nu\|\mathbf{z}\|(\hat{\mathbf{w}}_i - \boldsymbol{\alpha}_i) + \eta\|\mathbf{z}\|\mathbf{\Gamma}_i'^T\mathbf{r}_i) \quad \text{for } i = 1, 2, 3. \tag{88}$$

where $\boldsymbol{\alpha}_i$ is chosen as in Section 2.4. Considering that the errors can be denoted

$$\mathbf{r}_i = \mathbf{\Gamma}'_i\tilde{\mathbf{w}} + \boldsymbol{\tau}_i, \quad i = 1, 2, 3 \tag{89}$$

$$\delta_i = d_{\max} - \frac{\|\mathbf{r}_i\|\|\mathbf{z}_i\|}{\|\mathbf{z}_i\| + \varepsilon}, \quad i = 1, 2, 3 \tag{90}$$

where $\boldsymbol{\tau}_i$ are disturbances due to modeling limitations and perhaps differentiation, then using the notation

$$\boldsymbol{\tau} = \text{col}\{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \boldsymbol{\tau}_3\}, \tag{91}$$

$$\mathbf{\Gamma}' = \text{diag}\{\mathbf{\Gamma}'_1, \mathbf{\Gamma}_2, \Gamma'_3\}, \tag{92}$$

the Lyapunov derivative is

$$\dot{V} = -\mathbf{z}^T\mathbf{G}\mathbf{z} + \mathbf{z}\boldsymbol{\delta} + \|\mathbf{z}\|\tilde{\mathbf{w}}^T[\nu(\mathbf{w} - \boldsymbol{\alpha}) + \eta\mathbf{\Gamma}'^T\boldsymbol{\tau}] - \|\mathbf{z}\|\tilde{\mathbf{w}}^T(\nu\mathbf{1} + \eta\mathbf{\Gamma}'^T\mathbf{\Gamma}')\tilde{\mathbf{w}}, \tag{93}$$

which can be bounded as in the rigid case:

$$\dot{V} \leq -\|\mathbf{z}\|[\lambda_{\min}(\mathbf{G})\|\mathbf{z}\| - \delta_{\max} + \|\tilde{\mathbf{w}}\|\eta\lambda_{\max}(\mathbf{\Gamma}'^T\boldsymbol{\tau}) + \nu\|\tilde{\mathbf{w}}\|(\|\tilde{\mathbf{w}}\| - \|\tilde{\mathbf{w}} - \boldsymbol{\alpha}\|_{\max})], \tag{94}$$

and the stability result is achieved as in the rigid case.

4. CRAM

In the previous section, the theory has been developed for stable nueral-network control for a generalized associative memory. In fact, this theory is easily extended to multilayer perceptrons.[22] The important assumption is that the basis functions used are global in nature. However, the CMAC network contains strictly "hypercube" basic function. It can easily be demonstrated that this leads to instability when applied to a plant with elastic degree of freedom. In this section, a proposed hybrid of CMAC and RBF structures is described that is stable and also reproduces some of the desirable properties of the CMAC.

Let us first discuss why multilayer perceptrons may not be suitable for robotic control even though they are popular in the literature. Multilayered perceptrons (MLPs) learn relatively slowly and a great number of learning trials is needed to learn a robotic trajectory. A real robot moves slowly enough that long training times are impractical to accomplish. Qualifications to this are vibrational dynamics and motor dynamics where one can easily have many repetitions in a short time span. It is desirable in a robotic

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

system to be able to learn a trajectory in only a few learning trials. RBF (associative memory) networks have also been proposed in the literature.[23] The associative-memory approach learns much faster than the MLP approach however, RBF networks suffer from the *curse of dimensionality*. That is, the number of basis functions needed to approximate a function increases exponentially with the number of inputs. Thus for multilink robotic systems with full state input even careful placement of RBFs will result in a computationally intensive controller. To save memory, some sort of local functions must be used, as in the CMAC algorithm.

The reason that CMACs are unstable for flexible systems is that the state can input oscillate between discrete hypercube functions. Consider a CMAC that only has the desired coordinates as inputs, and those desired coordinates have no frequency content close to the natural frequencies of the system. In this case, the control will not excite the unstable dynamics. The restriction imposed is that only simple trajectories can be learned. However, this describes the type of task required for the vast majority of typical manipulator operations. For stability, the basis functions must also have knowledge of the currant state. This can be accomplished with the proposed CMAC-RBF Associative Memory (CRAM). Each CMAC cell has an RBF associated with it. The state variables are input to RBFs whose centers are placed dynamically at the point when the associated CMAC cell is activated. In this way, excellent function approximation is possible. A small disturbance is introduced over learning trials because each RBF center changes position each time a CMAC cell reentered. Overall, the resulting network is able to handle flexibilities while maintaining the efficient memory structure of the CMAC. The scheme is illustrated in Figure 3.

The basis functions are calculated as follows. In the CMAC, one hypercube cell per CMAC layer is uniquely indexed by the inputs $\{\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d\}$ which contains $3n$ components. The vector of inputs has a normalized position inside the cell, which can be denoted $\mathbf{h}_j$ on the jth layer, and this is the input to determine the value of the jth cell's basic function. The $3n$ components of $\mathbf{h}_j$ are denoted $h_{i,j} \ldots h_{3n,j}$, and the basic function value is

$$f_j = \prod_{i=1}^{3n} (h_{ij}^2 - 2h_{ij}^3 + h_{ij}^4). \tag{95}$$

The actual state input to $\boldsymbol{\Gamma}_i$ is $\mathbf{x} = \text{col}\{\mathbf{x}_1, \ldots, \mathbf{x}_{i+1}\}$ and is input to an RBF. The total CRAM basis function can thus be written

$$F_j = f_j \cdot \exp[-(\mathbf{x} - \mathbf{c})^T(\mathbf{x} - \mathbf{c})/(2\sigma_s^2)]. \tag{96}$$

where $\mathbf{c}$ is the center of the RBF, which has been dynamically placed near the trajectory as the corresponding CMAC cell was first activated. An example of this basis function for two inputs is drawn in Figure 4. The factor σ_s^2 is chosen to trade off accuracy and generalization. As a final step, the normalized basis functions denoted $\bar{F}_j$ are used. In terms of the adaptive control notation, the term $\boldsymbol{\Gamma}_1$ for a two link robot is of the form

$$\boldsymbol{\Gamma}_1 = \begin{bmatrix} \bar{F}_1 & \cdots & \bar{F}_k & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \bar{F}_1 & \cdots & \bar{F}_k \end{bmatrix}. \tag{97}$$

The result for a two dimensional input vector, without the normalization, is illustrated in Figure 3 where there are three layers of CMAC shown each with three coarse cells.

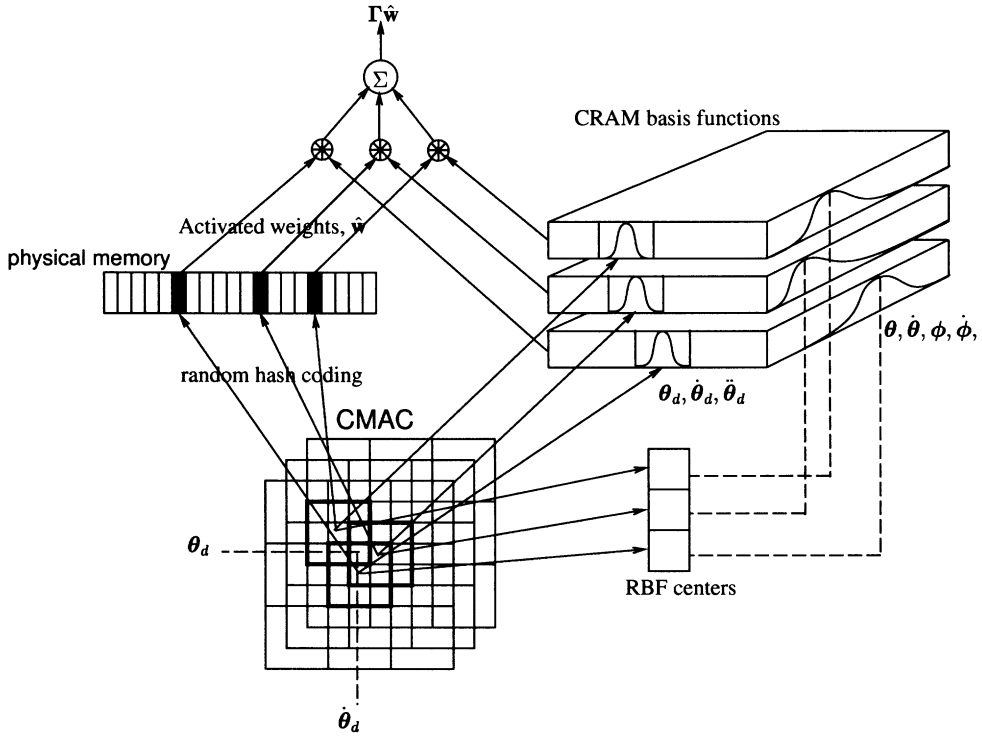

Fig. 3. CMAC-RBF associative memory (CRAM).

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

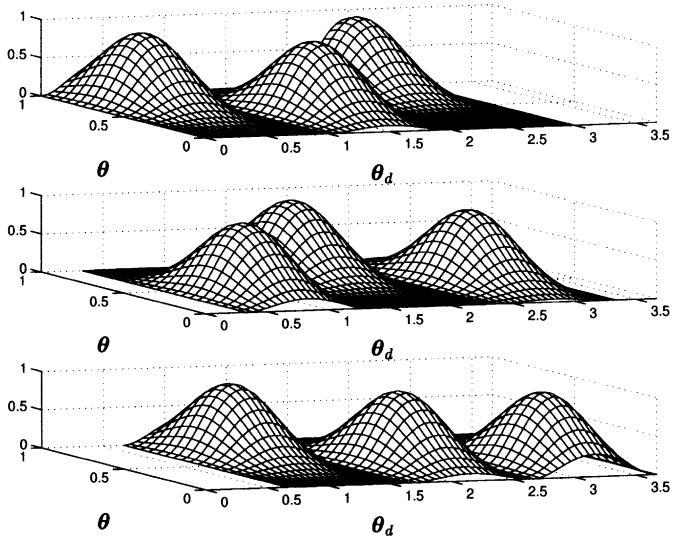

Fig. 4. CRAM basis function.

5. SIMULATION RESULTS

The robot model selected for numerical simulation was that of a two-link planar elastic-joint robot as in Figure 2, with physical parameters shown in Table I. The natural frequencies at $\theta_1 = \theta_2 = 0$ are 0.81 Hz and 1.02 Hz and the damping ratios are 0.0015. This results in a manipulator where the elastic deflections are significant and where vibrations take a long time to dissipate. The task in the simulation is to "catch" an object moving at 15cm/s in the y direction, and then come to an (almost) immediate stop as illustrated in Figure 5. This demands a matching of trajectories. The path of the object is not input as the desired trajectory. Rather the trajectory is planned in Cartesian coordinates in order to accelerate smoothly to the desired $\dot{y} = 15$cm/s velocity, and when $y = 75$cm is reached the stop is commanded in joint coordinates. This task is difficult in two ways. First, the initial desired acceleration, and the final commanded deceleration, are both quite large, so that it is difficult with a relatively small neural network to get an accurate dynamical model. This can lead to very poor performance in other schemes where learning occurs until some small error is reached. The second difficulty is that accuracy is required in the tip velocity as well as the tip position. In joint coordinates, this translates into fairly large, variable accelerations and velocities.

Table I. Manipulator physical properties

property	link 1	link 2
mass *kg*	1.0	1.0
tip mass *kg*	2.0	1.0
length *m*	1.0	1.0
	joint 1	joint 2
motor inertia $N \cdot m^2$	0.4	0.2
stiffness $(N \cdot m \cdot s)/rad$	10.0	5.0

The nonlinear effects are quite significant when the accuracy is the most important.

The modifications presented in this paper sequentially augment, in the order presented, resulting in five different control schemes. The labels A through E are used as in Table II. The control parameters were chosen as $\mathbf{\Lambda} + \mathbf{G}_1 = \mathbf{G}_2 = \mathbf{G}_3 = \text{diag}\{2, 2\}$, $\beta = 1.0$, $v = 1.0$, $\eta = 1.0$, $\sigma_1 = \sigma_2 = \sigma_3 = 10^{-6}$, $\sigma_s = 0.1$, $\varepsilon = 0.01$..

The objective was to obtain good performance with a small neural network. To this end, only 25 CMAC layers are used to learn each function, corresponding also to 25 activated weights that need to be integrated. The number of total weights is more because many functions must be learned. Each $\mathbf{\Gamma}_i$, for $i = 1, 2, 3$ has $n = 2$ outputs. In addition $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_3$ have desired outputs also at $\mathbf{\Gamma}_1'$ and $\mathbf{\Gamma}_3'$. thus the number of weights used was 50 for each output, $\mathbf{\Gamma}_1'$ in $\mathbf{\Gamma}_3'$ and 25 for each output in $\mathbf{\Gamma}_2$ for a total of 250 on-line weights for our two link robot. Note that for the CMAC algorithm, when learning is done in other parts of the state space more weights are stored in the physical memory, but no increased

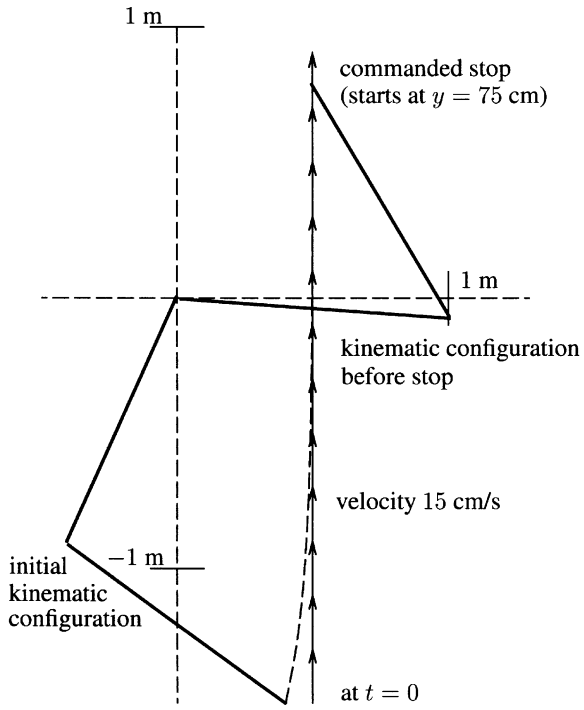

Fig. 5. Catching task schematic.

Table II. Controller simulations

Label	(Virtual) Control	Weight Update
A	unmodified (66)–(68)	unmodified (69)
B	optimal	unmodified
C	optimal	on-line learning
D	optimal and robust	on-line learning
E	optimal and robust (85)–(87)	on-line learning and modified robust (88)

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

Table III. CMAC cell sizes

Inputs	Cell Size Range
$\boldsymbol{\theta}_d$	0.4–3.0 *rad*
$\dot{\boldsymbol{\theta}}_d$	0.15–0.2 *rad/s*
$\ddot{\boldsymbol{\theta}}_d$	0.4–1.0 *rad/s*^{2}

on-line computational burden occurs. The cells sizes used in the CMAC are listed in Table III.

The learning curve in Figure 6 clearly shows that each performance modification in fact improves the performance. Note the dramatic improvement using the robust control term, included in control (D). This is because the neural network is incapable of learning the dynamics with much accuracy during the high acceleration and deceleration parts of the trajectory. However, the neural network does improve the performance during the "tracking phase" of the trajectory when low accelerations are needed. If the trajectory had low commanded accelerations throughout, the on-line learning would result in much improved performance on the learning curve as demonstrated in Macnab and D'Eleuterio.[13] A significant result is that the modified robust weight update, included in control (E), does improve the performance significantly over the entire trajectory. This allows performance improvement without more of the added torque chatter associated with the robust

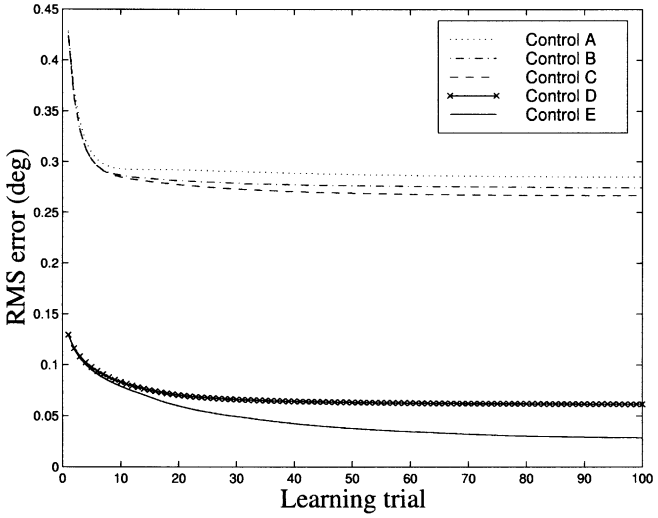

Fig. 6. Learning curve.

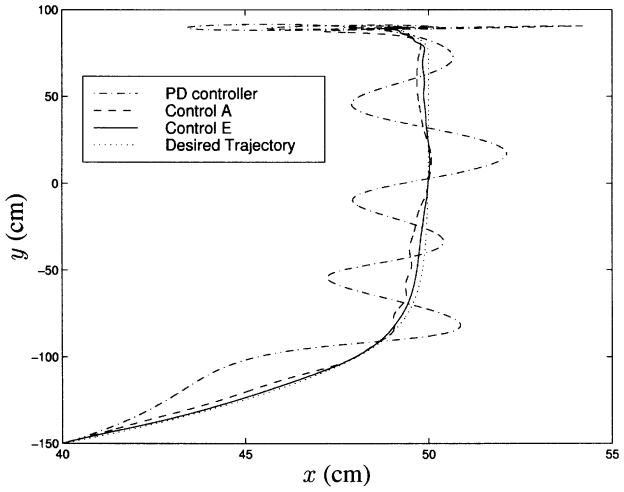

Fig. 7. First learning trial.

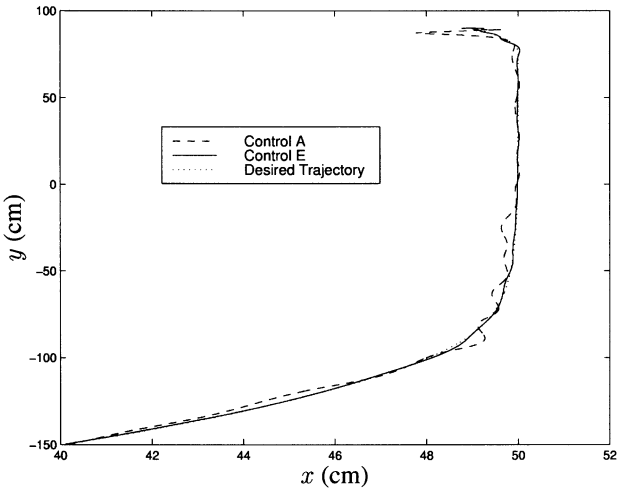

Fig. 8. 100th learning trial.

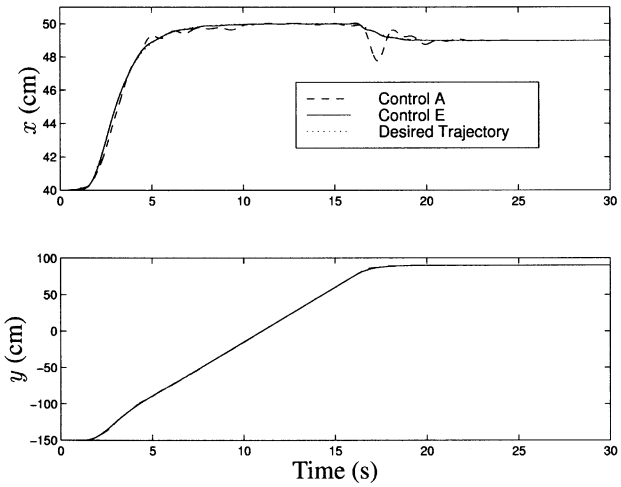

Fig. 9. Cartesian coordinates in time (100th trial).

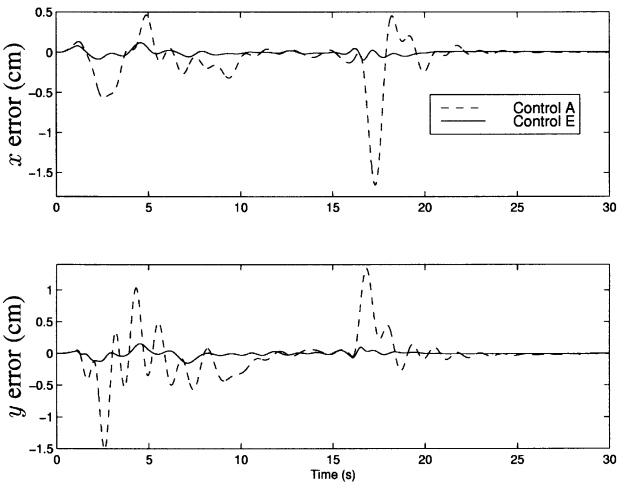

Fig. 10. Cartesian errors in time (100th trial).

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press

control term. The amount of performance improvement can be viewed in task space, for the 1st and 100th learning trials, as in Figures 7 and 8. The Cartesian coordinates and Cartesian errors in time can be viewed in Figures 9 and 10.

6. CONCLUSION

A neural network scheme has been presented for control of elastic-joint robots. This scheme is more practical to implement than other neural-network approaches ones owing to reduced training times and reduced on-line computational burden. A neural network architecture referred to as CMAC-RBF Associative Memory (CRAM) produces many of the properties of the CMAC algorithm in the elastic case. This results in stability, fast training, and reasonable on-line computations. To compensate for modeling limitations due to restricted size of the network, robust performance enhancements to both control torque and weight updates have been proposed. The estimate of the disturbance in the robust term comes directly from the error in the modeling of the current state. Their effect was demonstrated through simulations, and the robust control can result in excellent performance.

References

1. M. Krstic, I. Kanellakopoulos and P. Kokotovic, *Nolinear adaptive Control Design* (Wiley, New York, 1995).
2. S. Nicosia and P. Tomei, "A method to design adaptive controllers for flexible joint robots," *Proc. IEEE Int. Conf. Robotics and Automation* Nice France (1992) pp. 701–706.
3. D.M. Dawson, Z. Qu and J. Carroll, "Robust tracking of rigid-link flexible-joint electrically-driven robots," *Proc. 30th. Conf. Decision and Control* (1991) pp. 1409–1412.
4. B. Brogliato, R. Ortega and R. Lozano, "Global tracking controllers for flexible-joint manipulators: a comparative study," *Automatica* **31** (7) 941–956 (1995).
5. C.M. Kwan, F.L. Lewis, and Y.H. Kim, "Robust neural network control of flexible-joint robots," *Proc. 34th. Conf. Decision and Control* New Orleans (1995) pp. 1296–1301.
6. P.A. Ioannou, *Robust Adaptive Control* (Prentice-Hall, Upper Saddle River, NJ, 1996).
7. N.S. Narendra and A.M. Annaswamy, "A new adaptive law for robust adaptive control without persistance of excitation," *IEEE Trans. Aut. Control* **AC-32** 134–135 (1987).
8. J. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," *J. Dyn. Sys. Meas. Contr.* **97**, 220–227 (1975).
9. J. Albus, "Data storage in the cerebellar model articulation controller (CMAC)," *J. Dyn. Sys. Meas. Contr.* **97**, 228–233 (1975).
10. T.W. Millar, "Sensor-based control of robotic manipulators using a general learning algorithm," *J. Robot. Autom.* **RA-3**, (2), 157–165 (1987).
11. T.W. Millar, "Real-time dynamic control of an industrial manipulator using a neural-network based learning controller," *IEEE Trans. Robot. Autom.* **6**, (1), 1–8 (1990).
12. P.W. Graham and G.M.T. D'Eleuterio, "A neural network paradigm for robotic control," *Can. Aero. Space J.* **37**, (1), 17–26 (1991).
13. C.J.B. Macnab and G.M.T. D'Eleuterio, "Stable, on-line learning using CMACs for neuroadaptive tracking control of flexible-joint manipulators," *Proc. IEEE Int. Conf. robotics and Automation* Leuven, Belgium (1998) **Vol. 1**, pp. 511–517.
14. J. Slotine and W. Li, "On the adaptive control of robot manipualtors," *J. Robotics Research* **6**, (3), 49–59 (1987).
15. K.S. Narendra and A.M. Annaswamy, *Stable Adaptive Systems* (Englewood Cliffs, NJ, Prentic-Hall, 1989).
16. S. Gutman, "Uncertain dynamics systems–Lyapunov min-max approach," *IEEE Transactions on Automatic Control*. **24**, 437–443 (1979).
17. Z. Qu and D.M. Dawson, *Robust Tracking Control of Robot Manipulators* (IEEE Press, New York, 1996).
18. M.J. Corless and G. Leitmann, "Continuous state feedback guaranteeing uniform ultimate boundedness for uncertain dynamic systems," *J. Dyn. Sys. Meas. Contr.* **26**, 1139–1144 (1981).
19. R.A. Freeman and P.V. Kokotovic, *Robust Nonlinear Control Design: State-Space and Lyapunov Techniques*, (Birkhauser, Boston, 1996).
20. J. Slotine and W. Li, "Adaptive robot control: A New Perspective," *Proc 26th IEEE Conf. Decision and Control.* Los Angeles (1987) **Vol, 1**, 192–198.
21. M.W. Spong and M. Vidyasager, *Robot Dynamics and Control* (John wiley and Sons, New York, (1989).
22. M. Ciliz and C. Isik, "Stability and convergence of neurologic model based robotic controllers," *Proc. IEEE Int. Conf. Robotics and Automation*. Nice France (1992) pp. 2051–2055.
23. R.M. Sanner, and J. Slotine, "Guassian networks for direct adaptive control," *IEEE Trans. Neural Networks.* **3**, 837–863 (1992).

https://doi.org/10.1017/S0263574799002155 Published online by Cambridge University Press