JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 27, No. 5, September–October 2004

Neural-Network-Based Design of Optimal Controllers for Nonlinear Systems

Nilesh V. Kulkarni* and Minh Q. Phan†
Dartmouth College, Hanover, New Hampshire 03755

A neural-network-based methodology for the design of optimal controllers for nonlinear systems is presented. The overall architecture consists of two neural networks. The first neural network is a cost-to-go function approximator (CTGA), which is trained to predict the cost to go from the present state of the system. The second neural network converges to an optimal controller as it is trained to minimize the output of the first network. The CTGA can be trained using available simulation or experimental data. Hence an explicit analytical model of the system is not required. The key to the success of the approach is giving the CTGA a special decentralized structure that makes its training relatively straightforward and its prediction quality carefully controlled. The specific structure eliminates many of the uncertainties often involved in using artificial neural networks for this type of application. Validity of the approach is illustrated for the optimal attitude control of a spacecraft with reaction wheels.

I. Introduction

ARTIFICIAL neural networks have been investigated extensively in the optimal control of nonlinear systems. Control architectures known as adaptive critic designs (ACD) have been proposed for the optimal control problem.[1–3] ACD are based on the forward dynamic programming approach to optimization. The basic architecture consists of a critic that models the cost-to-go function and a controller. Both structures are parameterized using neural networks. These two functions are trained simultaneously and each of them depends on the fidelity of the other to get trained properly. This could make the training of the overall system particularly challenging. An interesting solution approach has been presented recently, where the critic and the controller are pretrained using linear models over the region of operation of the system and an algebraic training procedure is employed in the initialization of the neural networks.[4]

In our previous work, a modified parametric optimization approach was developed to generate optimal controllers in both state feedback form and dynamic output feedback form for linear systems.[5] In the present work, we generalize the approach to nonlinear systems. Parametric optimization imposes the form of the controller in advance and the controller parameters to optimize the performance measure are found. If the controller parameterization is done via neural networks, then given their universal function-approximating capability, the true optimal controller can in theory be captured.[6,7] One objective of our research is to remove many of the uncertainties associated with training a neural network architecture that results in an optimal controller. The key to the success of our method is to give the neural network a very special structure that permits tight control over its prediction quality during training. The special structure also makes it possible for each subsystem of the overall network to be trained independent of other subsystems. The issue of interdependency among various portions of the overall network during training, encountered in the basic adaptive critic designs, is therefore removed. For large multivariable nonlinear systems, reaching the global optimum when working with gradient-based optimization approaches can never be guaranteed. We also illustrate a hybrid addition to the method where the optimal nonlinear controller gets the system close to the desired state and then a linear controller gets it exactly to the desired state.

In addition to being motivated by ACD, our work is also an application of concepts put forth in adaptive and predictive control.[8–12] Section II outlines the overall control architecture. Sections III and IV present the details of training of the cost-to-go function neural network and the controller neural network, respectively. Section V presents an application of the method for the attitude control of a spacecraft using reaction wheels. The final concluding remarks are presented in Section VI.

II. Overall Neural Network Architecture

As a starting point, we briefly motivate the modified parametric optimization approach developed in our previous work.[5]

A. Modified Parametric Optimization

For a given system

$$x(k+1) = f[x(k), u(k)] \tag{1}$$

the typical parametric optimization approach assumes a form for the controller

$$u(k) = u[x(k), G] \tag{2}$$

The structure of the function u is assumed and the parameters G are found to minimize a chosen cost-to-go function such as

$$V(k) = \frac{1}{2}\sum_{i=1}^{r}[x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1] \tag{3}$$

For the infinite-horizon optimal control problem, the upper limit of the summation in the cost-to-go function should be infinity. Here, it is replaced by a finite value r that can be thought of as the order of approximation of the infinite-horizon cost-to-go function. By the principle of optimality, optimizing the cost-to-go function is equivalent to optimizing the cumulative cost. This is the mechanism by which receding-horizon model predictive control handles the optimal control problem. Thus it is expected that as r tends to infinity, the resultant control will converge to the truly optimal control solution. The parameters in G are found by imposing the necessary condition

$$\frac{\partial V}{\partial G} = 0 \tag{4}$$

Presented as Paper 2002-4664 at the AIAA Guidance, Navigation and Control Conference, Monterey, CA, 5 August 2002; received 8 May 2003; revision received 5 December 2003; accepted for publication 14 December 2003.

*Visiting Student, Thayer School of Engineering; currently Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544; nkulkarn@princeton.edu. Student Member AIAA.

†Associate Professor, Thayer School of Engineering; Minh.Q.Phan@dartmouth.edu. Member AIAA.

Solving the problem in this manner leads to a highly nonlinear algebraic-optimization problem with many local minima even for a linear system. From the point of view of the neural networks for optimal control, which must be trained iteratively, this solution approach is highly undesirable. The modified parametric optimization approach introduces additional unknowns to simplify the solution of the unknown parameters.[5] Instead of formulating the control law as given in Eq. (2), the modified approach formulates the control law as

$$\begin{aligned} u(k) &= u_1[x(k), G_1] \\ u(k+1) &= u_2[x(k), G_2] \\ &\vdots \\ u(k+r-1) &= u_r[x(k), G_r] \end{aligned} \tag{5}$$

The present and future control values are given by the nonlinear functions $u_1, u_2, \ldots, u_r$ of the present state and the vectors of parameters $G_1, G_2, \ldots, G_r$. We can write the present as well as future control inputs as functions of the present state as long as we assume the existence of a closed-form state feedback formulation for the controller. For example, the control $u(k+1)$ is a feedback function of the state $x(k+1)$, which, given Eq. (1), is a function of $x(k)$ and $u(k)$. Therefore, $u(k+1)$ is a function of the state $x(k)$. Formulations of the further control values as functions of the present state follows similar recursive argument. Future control values can therefore be presented as functions of the present state information. So instead of solving for one set of parameters G, we now solve for the expanded set of controller parameter vectors $G_1, G_2, \ldots, G_r$. In theory, these unknown parameters are found by imposing the conditions

$$\frac{\partial V}{\partial G_1} = 0, \qquad \frac{\partial V}{\partial G_2} = 0, \qquad \cdots \frac{\partial V}{\partial G_r} = 0 \tag{6}$$

$$\begin{aligned} G_2 &= G_2(f, G_1) \\ G_3 &= G_3(f, G_2) \\ &\vdots \\ G_r &= G_r(f, G_{r-1}) \end{aligned} \tag{7}$$

The additional conditions in Eqs. (7) are due to the fact that the extra gains are no longer independent, but are related through the system equations. It was, however, found that because the state $x(k)$ enters Eq. (6), by simply varying $x(k)$ arbitrarily, we will have a complete set of conditions from which to find the expanded set of gains without having to invoke Eq. (7).[5] Not having to include Eq. (7) in the solution technique is a major beneficial feature, because it is nonlinear even for a linear system. Furthermore, these conditions also involve the system model. Therefore, using Eq. (7) would also prevent the development of a data-based approach that does not require an explicit system model.

It was established in our previous work that for a linear system, the optimal control gain can be found by solving two sets of linear equations, one to identify the cost-to-go function, and the other to obtain the optimal controller gain. This solution approach is also data-based in that it can be carried out from available input–state or input–output data without needing an explicit model of the system in standard form. The modified parametric optimization approach is now used to formulate a general form of the neural-network control architecture. The overall architecture consists of two neural networks. One neural network is used as the cost-to-go function approximator (CTGA), and the other is used as the controller.

B. Neural Network CTGA

Figure 1 illustrates the CTGA neural network. Given the simulation or actual data of the system, for different starting values of the index k, the values of the states $x(k+1)$ through $x(k+r)$ and inputs $u(k)$ through $u(k+r-1)$ can be collected. From these values the true value of $V(k)$ can be computed using Eq. (3).

The states of the system $x(k+1)$ through $x(k+r)$ are functions of the state of the system $x(k)$ and the inputs $u(k)$ through

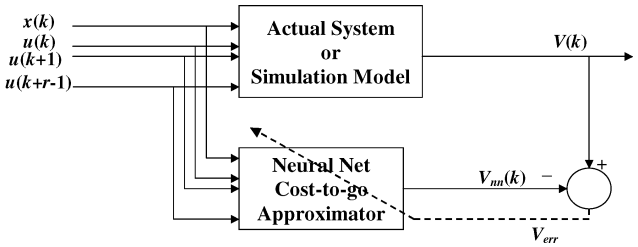


Fig. 1 Training the cost-to-go approximator network.

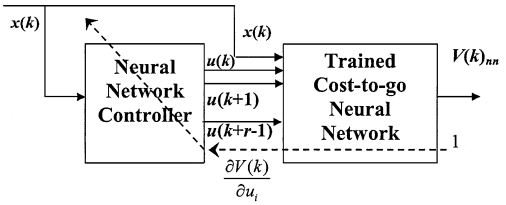


Fig. 2 Neural-network controller training.

$u(k+r-1)$, as discussed in the preceding section. The cost-to-go function $V(k)$ can therefore be modeled as a function of the state $x(k)$ and the inputs $u(k)$ through $u(k+r-1)$. A neural-network CTGA is thus formulated to take these as its inputs and provide the cost-to-go function estimate $V(k)$ as its output. The details of training the CTGA network are given in Sec. III.

C. Neural Network Controller

The neural network controller (NNC) can be parameterized according to Eqs. (5). The controller takes the state of the system as its input and provides the r control values $u(k)$ through $u(k+r-1)$. The outputs of the NNC, along with $x(k)$, are fed to the CTGA as described before. The NNC is trained so that for a range of states $x(k)$, the outputs of the NNC fed to the CTGA minimize the cost-to-go function estimate (see Fig. 2). The details of the structure of the NNC and the training procedure are reserved for Sec. IV.

The NNC and the CTGA form the overall control architecture. It can be noted that the combined network of the NNC and the CTGA looks similar to the critic network used in the heuristic dynamic programming formulation of the adaptive critic design.[3] In the present formulation, the CTGA network can be trained independent of the controller network using the available system data, and then used for training the controller. This form of decoupled training avoids the issue of having to train both networks simultaneously, as in the adaptive critic designs.

III. Cost-to-Go Approximator Network

The CTGA network proposed in the previous section has a large input space because it takes the state $x(k)$ and the input values $u(k)$ through $u(k+r-1)$ as its inputs. In practice, presenting the network with a broad range of physically realizable test inputs still likely leaves the network untrained over a significant portion of the input space. The enormousness of the input space is a typical training problem. An incompletely trained CTGA network will surely result in failure of the controller-network-training step. This is a very critical issue to overcome. A well-trained CTGA network is the most important factor in getting a successful nonlinear optimal controller. This problem leads us to give the CTGA a specific structure that facilitates its training.

Figure 3 presents the general outline of the proposed structure of the CTGA network. We introduce subnets 1 through r that correspond to the single- and multistep predictors. Each of these subnets corresponds to a two-layer (sigmoid–linear) neural network. Thus subnet i takes the state of the system $x(k)$ and the input values $u(k)$ through $u(k+i-1)$ as its inputs and provides the output $x(k+i)$. The outputs of these subnets, $x(k+1)$ through $x(k+r)$, along with the input values $u(k)$ through $u(k+r-1)$ are fed to a layer of squared neurons to compute the quadratic cost-to-go value $V(k)$.

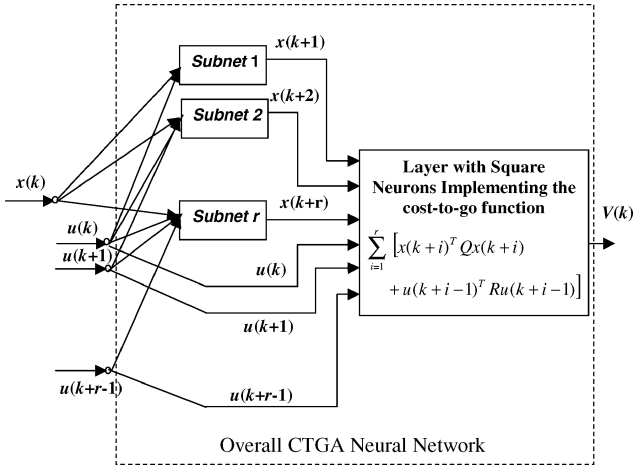


Fig. 3 CTGA Internal architecture.

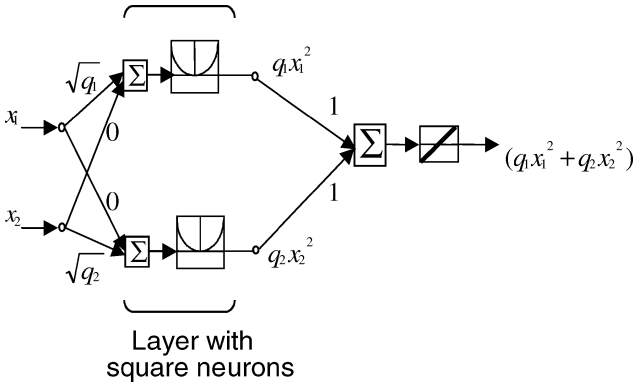


Fig. 4 Weighted quadratic product.

The weighting matrices Q and R are embedded in this layer. Figure 4 presents a simple example of producing a weighted quadratic product using such a layer of neurons. The presence of the quadratic layer helps to ensure the positive definiteness of the cost-to-go function $V(k)$, which is important in the next step of training the controller network. It should be noted that any functional form can be implemented using specific function neurons. There are two advantages of this fixed cost-to-go function layer. One is that there is no approximation involved in the calculating the cost-to-go function once the state and control predictions are available. The second is an important practical consideration involved in any control design. In order to get acceptable results, we typically have to tune the Q and R weighting matrices. This can be done easily with this fixed layer of neurons instead of having these values imbedded in the neural-network training.

Subnets 1 through r can be trained separately using the available system data. This ability to train the r subnets individually leads us to a CTGA network that is well trained overall. We note that with the proposed network structure we once again run into the problem of a large input space for the higher order subnets. For example, the subnet r receives all of the inputs presented to the CTGA network and therefore poses a significant challenge for training. With the subnet structure, there is a way to get around this difficulty. In practice the subnets can be trained up to a certain order $\hat{r}$ for which large input dimensionality is not an issue. These $\hat{r}$ subnets can then be stacked together to produce the higher order subnets. Each of these subnets can be well trained and tested for its prediction quality based on available data. Figure 5 gives an example of an order-10 CTGA network built with subnets with orders 1–5. This strategy allows systematic contruction of a high-quality CTGA network. Training the subnets up to an order $\hat{r}$ and then assembling them together, along with the fixed-cost-function-implementing layer using functional neurons, constitutes the training of the CTGA.

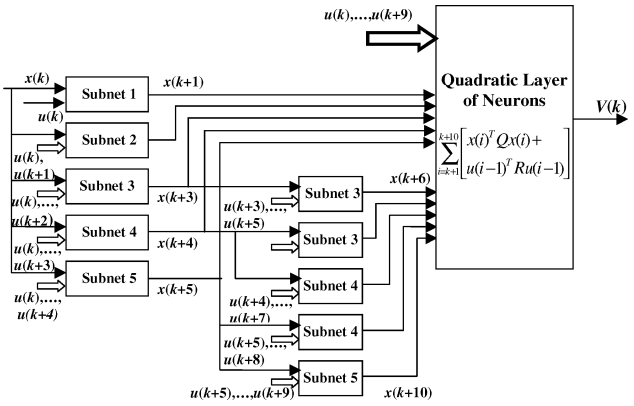


Fig. 5 Implementation of the CTGA of order $r = 10$, using trained subnets of orders 1–5.

IV. Neural Network Controller

In this section we discuss the specifics of the neural-network-controller parameterization and training.

A. Neural Network Structure

The neural network controller is modeled to follow Eqs. (5). Some practical considerations lead to different possible parameterizations of the controller neural network. The biggest consideration is controlling the frequency content of the control profile $u(k)$ through $u(k+r-1)$ that is produced by the controller. Subnet i takes controls $u(k)$ through $u(k+i-1)$ as its inputs, along with state $x(k)$, to predict state $x(k+i)$. The subnets are trained using input–state data where the inputs are chosen so that they excite the appropriate modes of the system.

Ideally the sampling interval Δt should be as small as possible to capture the fastest dynamics of the system and at the same time as large as possible to minimize the number of inputs r, and hence the size of the CTGA network. This is an important issue and problem-specific. A choice of Δt essentially places a limit on the frequency content of the output of the controller network. A control profile that has its frequency content beyond the values for which the subnets are trained deteriorates the subnet prediction quality. This in turn can lead to erroneous results in the controller optimization procedure.

One way to impose this is to have the neural network controller output the coefficients of a set of basis functions that combine to give a smooth control signal,

$$u(k+i) = \sum_{j=1}^{N} \alpha_j[x(k)]\phi_j(i), \qquad i = 0, 1, \ldots, r-1 \tag{8}$$

Given the state of the system $x(k)$, the neural network controller outputs the coefficients α_j, $j = 1, \ldots, N$, for N basis functions. The choice of the basis functions remains arbitrary. Another option is to impose an artificial control-derivative weighting on the cost-to-go function. So the original cost-to-go function given by Eq. (3) is appended,

$$V(k) = \sum_{i=1}^{r} [x(k+i)^T Q x(k+i) + u(k+i-1)^T R u(k+i-1)] + \sum_{i=1}^{r-1} [u(k+i) - u(k+i-1)]^T R_{\text{der}} [u(k+i) - u(k+i-1)] \tag{9}$$

This added term in the cost-to-go function will not alter the original optimal profile much if the control profile is sufficiently smooth. However, it will substantially increase the cost-to-go function for nonsmooth profiles, leading the optimization procedure away from such solutions. The choice of the weighting term R_{der} provides a design choice for the control designer. The choice of a particular method between the two discussed can remain problem-specific. In the illustration shown in the results section we use the second method with an appropriate value for the design-weighting term R_{der}.

Finally, it is important to point out that if random input training data provide good prediction quality for the trained subnets then the constraints are not needed. In most realistic scenarios, however, the controller will need to be trained using experimental data that would rarely have a random profile.

Because we impose the frequency constraint indirectly through the cost function, the controller neural network structure can be chosen as the standard two-layer network with the hidden layer neurons having sigmoid activation functions and the output layer neurons having linear activation functions. Another choice in the structure of the neural network controller arises in imitating Eqs. (5). To follow Eqs. (5), strictly, the neural network, controller can be modelled as r independent two-layer networks, each of which takes $x(k)$ as its input and outputs $u(k+i), i=1,\ldots,r$. The other choice is to have a single fully connected two-layer network output all r control outputs. This again is a design choice that remains problem-specific.

B. Neural Network Controller Training

To train the controller network, whatever the chosen structure, a combined network is formed that consists of the controller network and the CTGA network (see Fig. 2). The controller part of the network has the internal structure as chosen, and the CTGA part of the network has the structure shown in Fig. 5.

The combined network takes the state of the system as its input and gives the cost-to-go estimate $V(k)$ as its output. The input space of this network is therefore comparatively small. To optimize the neural network controller, the combined network is trained as a whole. However, the weights and biases corresponding to the CTGA part of the network are held fixed, and only the controller part of the network is updated. To accomplish its training, the combined network is provided with a randomized set of the states $x(k)$ over the desired training range. For all of these inputs, the training is configured so that the desired output of the combined network is required to be zero. The positive definite quadratic layer in the CTGA part of this network makes sure that the cost-to-go function is always positive for nonzero states and controls. Thus, when the desired outputs for all of the training inputs are set to zero, the training ends up finding a set of weights and biases that minimizes the cost-to-go function $V(k)$ for all of the given states $x(k)$. This is indeed the desired result. The training minimizes $V(k)$ by training the controller part of the combined network to produce an optimal controller.

C. Hybrid Nonlinear–Linear Control

For large multivariable nonlinear systems, there is always the possibility that a gradient-based optimization procedure will not succeed in reaching the global minimum. A typical approach is to vary the starting points in the parameter search space. If the nonlinear controller produces a solution that is close to the desired terminal state, then it is possible to improve the performance by switching to a linear controller. In this section we describe such a hybrid nonlinear–linear control approach.

Subnet 1 in the CTGA network corresponds to the one-step-ahead neural network model of the system,

$$x(k+1) = f_{\mathrm{NN}}[x(k), u(k)] \tag{10}$$

Given the present state $x(k)$ and the control $u(k-1)$ that has been effectively used till time index k, we can use backpropagation through the neural network model to get

$$A_{\mathrm{NN}}(k) = \left.\frac{\partial f_{\mathrm{NN}}}{\partial x}\right|_{[x(k),u(k-1)]}, \qquad B_{\mathrm{NN}}(k) = \left.\frac{\partial f_{\mathrm{NN}}}{\partial u}\right|_{[x(k),u(k-1)]} \tag{11}$$

A discrete-time state-dependent linear–quadratic regulator can be designed using the state-dependent system matrices $A_{\mathrm{NN}}(k)$ and $B_{\mathrm{NN}}(k)$ and the cost-to-go function used in the nonlinear control design without the control derivative weighting. This procedure can be carried out at every time step to get the linear optimal control

$$u(k) = G_{\mathrm{dlqr}}(k)x(k) \tag{12}$$

This procedure need not be used if acceptable performance is achieved using the nonlinear design alone.

V. Implementation Results

The proposed control architecture is illustrated for the optimal attitude control of a spacecraft.

A. System Description

The spacecraft is controlled using three reaction wheels, each along its principal axis. The equations of motion of the spacecraft are given in the inertial frame of reference. We also include the actuator dynamics of the reaction wheel.

The attitude kinematics is given using quaternions:

$$\dot{q} = \frac{1}{2}\begin{bmatrix} 0 & \omega_1 & \omega_2 & \omega_3 \\ -\omega_1 & 0 & \omega_3 & -\omega_2 \\ -\omega_2 & -\omega_3 & 0 & \omega_1 \\ -\omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix} q \tag{13}$$

Here $q=[q_1\ \ q_2\ \ q_3\ \ q_4]^T$ is the quaternion taking a vector from the inertial to the body frame; $\omega=[\omega_1\ \ \omega_2\ \ \omega_3]^T$ is the body-frame angular velocity of the spacecraft. We use the quaternions to represent the attitude information because we do not restrict the angular deviations to be small.

The spacecraft dynamics is given by the equation

$$I_{\mathrm{SC}}\dot{\omega} = -\omega \times [I_{\mathrm{SC}}\omega + I_{\mathrm{RW}}(\omega + \omega_{\mathrm{RW}})] - T_{\mathrm{reaction}} \tag{14}$$

Here I_{SC} is the spacecraft inertia matrix and I_{RW} is the scalar moment of inertia of the reaction wheel along its axis of rotation. $\omega_{\mathrm{RW}}=[\omega_{\mathrm{RW1}}\ \ \omega_{\mathrm{RW2}}\ \ \omega_{\mathrm{RW3}}]^T$ is the vector of angular velocities of the three reaction wheels. Further, all of the reaction wheels have the same value of moment of inertia I_{RW} along their rotation axes. T_{reaction} is the reaction torque vector given by the three reaction wheels.

The dynamics of the reaction wheel is governed by the equation

$$I_{\mathrm{RW}}\dot{\omega}_{\mathrm{RW}} = -I_{\mathrm{RW}}\dot{\omega} + T_{\mathrm{reaction}} \tag{15}$$

The input to each reaction wheel is the specified bus voltage. The voltage is bounded on the positive and negative sides by the maximum bus voltage. The input voltage gives rise to a current given by

$$i = (v - k_T\omega_{\mathrm{RW}})/r \tag{16}$$

$v=[v_1\ \ v_2\ \ v_3]^T$ is the vector of voltage inputs. The term $k_T\omega_{\mathrm{RW}}$ accounts for the back emf in the wheels and r is the resistance of the bus. This resulting input current is bounded on the positive and negative sides by the maximum bus current. The reaction torque is given by

$$T_{\mathrm{reaction}} = k_T i - k_\omega\omega_{\mathrm{RW}} \tag{17}$$

k_T and k_ω are constants for the wheel. The above reaction torque neglects stiction and coulomb friction in the wheels, which may need closer attention.

This nonlinear model can be written in the standard form

$$\dot{x} = f(x,u) \tag{18}$$

with $x=[q^T\ \ \omega^T\ \ \omega_{\mathrm{RW}}^T]^T$ the system states and $u=[v_1\ \ v_2\ \ v_3]^T$ the input voltages to the three reaction wheels. The constants used in the spacecraft model are $I_{\mathrm{sc}_{11}}=1000\ \mathrm{kg\cdot m^2}$, $I_{\mathrm{sc}_{22}}=800\ \mathrm{kg\cdot m^2}$, $I_{\mathrm{sc}_{33}}=1500\ \mathrm{kg\cdot m^2}$, $I_{\mathrm{RW}}=0.0398\ \mathrm{kg\cdot m^2}$, $k_T=0.0329\ \ \mathrm{V\cdot s}$, $k_\omega=4.93\times10^{-5}\ \ \mathrm{N\cdot m\cdot s}$, $r=1.8856\Omega$, $|i|_{\max}=3.47A$, $k_T=0.0329\ \mathrm{V\cdot s}$, and $|r_{in}|_{\max}=28\ \mathrm{V}$. This model is provided by Princeton Satellite Systems.[13]

The data are generated by integrating the equations using a fourth-order Runge–Kutta solver. The sampling interval is chosen to be 2 s. The discretization frequency is chosen based on preliminary experimentation with the system at different excitation frequencies. The inputs to the system are produced by taking a random signal filtered through a discrete low-pass Butterworth filter. This removes the high-frequency content of the signal, which does not significantly excite the system. It is also assumed that all the states can be measured. This discrete form of the input–state history is then used to train the CTGA and then the NNC. The subnets are trained using the Levenberg–Marquardt algorithm and the NNC is trained using the RPROP algorithm.[14,15]

B. Cost-to-Go Function Formulation

The cost-to-go function is given by

$$V(k)=\sum_{i=1}^{r}\begin{bmatrix} q_e'(k+i)^T Q_{\text{quart}} q_e'(k+i) \\ \quad +\omega(k+i)^T Q_\omega \omega(k+i) \\ \quad +\omega_{\text{RW}}(k+i)^T Q_{\text{RW}} \omega_{\text{RW}}(k+i) \\ \quad +v_{in}(k+i-1)^T R v_{in}(k+i-1) \end{bmatrix}$$

$$+\sum_{i=1}^{r-1}[v_{in}(k+i)-v_{in}(k+i-1)]^T R_{\text{der}}[v_{in}(k+i) - v_{in}(k+i-1)] \tag{19}$$

When the desired and the actual attitudes coincide, the quaternion error $q_e(k)$ equals $q_0=[1\ 0\ 0\ 0]^T$. $q_e'(k)$ is the vector of the last three elements of the quaternion error. The quarternion error, $q_e(k)$, is given by $A_{\text{des}}q(k)$ (Ref. 16). The matrix A_{des} is given by

$$A_{\text{des}}=\begin{bmatrix} q_{1\text{des}} & q_{2\text{des}} & q_{3\text{des}} & q_{4\text{des}} \\ -q_{2\text{des}} & q_{1\text{des}} & q_{4\text{des}} & -q_{3\text{des}} \\ -q_{3\text{des}} & -q_{4\text{des}} & q_{1\text{des}} & q_{2\text{des}} \\ -q_{4\text{des}} & q_{3\text{des}} & -q_{2\text{des}} & q_{1\text{des}} \end{bmatrix} \tag{20}$$

where the desired attitude is specified with the desired quaternion $q_{\text{des}}=[q_{1\text{des}}\ q_{2\text{des}}\ q_{3\text{des}}\ q_{4\text{des}}]^T$. The cost function uses $q_e'(k)$ instead of $q_e(k)$ because the attitude is represented using a four-variable quaternion with a nonlinear constraint between the variables $q_e(k)^T q_e(k)=1$. Q_{quart}, Q_ω, Q_{RW}, R_{RW}, and R_{der} are the positive weightings for the respective terms in the cost-to-go function. In this illustration we choose to reorient the spacecraft so that the body axes of the spacecraft are aligned with the inertial axes. This corresponds to $q_{\text{des}}=[1\ 0\ 0\ 0]^T$ and the matrix A_{des} reduces to an identity matrix. The weighting parameters used in the cost-to-go function are $Q_{\text{quart}}=\text{diag}([40\ \ 40\ \ 40])$, $Q_w=\text{diag}([10{,}000\ \ 10{,}000\ \ 10{,}000])$, $Q_{\text{RW}}=\text{diag}([10^{-5}\ \ 10^{-5}\ \ 10^{-5}])$, $R_{\text{RW}}=\text{diag}([10^{-3}\ \ 10^{-3}\ \ 10^{-3}])$, and $R_{\text{der}}=\text{diag}([0.2\ \ 0.2\ \ 0.2])$.

The CTGA is assembled with subnets 1 through 4. Higher order prediction is achieved by cascading these subnets as illustrated in Fig. 5. There remains the issue of choosing the order of prediction r. It is observed that this depends on the system dynamics. The order of prediction r or the prediction horizon needs to be chosen so that the transient response can be sufficiently captured. Having the order of prediction greater than this value adds little benefit while substantially increasing the computational burden. In the present problem, $r=32$ is sufficient. A more detailed discussion on this issue can be found in Ref. 17.

The control input has three elements corresponding to the voltage inputs to the three reaction wheels. The neural-network controller is parameterized as three two-layer (sigmoid–linear layer) networks where each two-layer structure outputs the voltage input profile for a reaction wheel over the chosen prediction horizon r. For a given state $x(k)$, the first control value among the r values is the input to the system.

C. Implementing Hybrid Nonlinear–Linear Control

It is observed that the nonlinear optimal controller gets the spacecraft within a degree of the required attitude. We now implement the hybrid nonlinear–linear control as explained in Sec. IV.

In its present formulation, the linear system will be uncontrollable because the attitude is represented using a four-variable quaternion with a nonlinear constraint between the variables. However, close to the desired attitude, the first component of the quaternion error decouples itself as the uncontrollable variable. The analytical model described is only used to generate the simulation data and the controller design uses these data to get the optimal controller. However, to illustrate some issues, we can take a close look at the analytical model. When the attitude error tends to zero, $q_{1e}(k)\approx 1$ and the kinematic equations reduce to

$$\dot{q}_{1e}=0, \qquad \dot{q}_{2e}=-\tfrac{1}{2}w_1, \qquad \dot{q}_{3e}=-\tfrac{1}{2}w_2, \qquad \dot{q}_{3e}=-\tfrac{1}{2}w_3 \tag{21}$$

Thus, if we were designing a linear controller knowing the model of the system, the equation for q_{1e} could be neglected and for the other dynamic equations, q_{1e}can be substituted as 1, and these can be used to obtain a linear controller. Since we have the trained subnet 1, which corresponds to the one-step-ahead neural-network model, we can use backpropagation to get

$$\begin{bmatrix} q(k+1) \\ \omega(k+1) \\ \omega_{\text{RW}}(k+1) \end{bmatrix}=A_{\text{NN}}(k)\begin{bmatrix} q(k) \\ \omega(k) \\ \omega_{\text{RW}}(k) \end{bmatrix}+B_{\text{NN}}(k)\begin{bmatrix} v_{\text{in1}}(k) \\ v_{\text{in2}}(k) \\ v_{\text{in3}}(k) \end{bmatrix} \tag{22}$$

To look at the error dynamics, we use the fact that $q_e(k)=A_{\text{des}}q(k)$ to get

$$\begin{bmatrix} q_e(k+1) \\ \omega(k+1) \\ \omega_{\text{RW}}(k+1) \end{bmatrix}=\begin{bmatrix} A_{\text{des}} & 0_{4x3} & 0_{4x3} \\ 0_{3x4} & I & 0_{3x3} \\ 0_{3x4} & 0_{3x3} & I \end{bmatrix}A_{\text{NN}}(k)$$

$$\times\begin{bmatrix} A_{\text{des}} & 0_{4x3} & 0_{4x3} \\ 0_{3x4} & I & 0_{3x3} \\ 0_{3x4} & 0_{3x3} & I \end{bmatrix}^{-1}\begin{bmatrix} q_e(k) \\ \omega(k) \\ \omega_{\text{RW}}(k) \end{bmatrix}$$

$$+\begin{bmatrix} A_{\text{des}} & 0_{4x3} & 0_{4x3} \\ 0_{3x4} & I & 0_{3x3} \\ 0_{3x4} & 0_{3x3} & I \end{bmatrix}B_{\text{NN}}(k)\begin{bmatrix} v_{\text{in1}}(k) \\ v_{\text{in2}}(k) \\ v_{\text{in3}}(k) \end{bmatrix}$$

Neglecting the dynamics of $q_{1e}(k)$, we can write

$$\begin{bmatrix} q_e'(k+1) \\ \omega(k+1) \\ \omega_{\text{RW}}(k+1) \end{bmatrix}=A'_{\text{NN}}(k)\begin{bmatrix} q_e'(k) \\ \omega(k) \\ \omega_{\text{RW}}(k) \end{bmatrix}+B'_{\text{NN}}(k)\begin{bmatrix} v_{\text{in1}}(k) \\ v_{\text{in2}}(k) \\ v_{\text{in3}}(k) \end{bmatrix} \tag{23}$$

where $q_e'(k)=[q_{2e}(k)\ \ q_{3e}(k)\ \ q_{4e}(k)]^T$. Equation (23), along with the cost-to-go function given by Eq. (19) without the control derivative weighting term is used to design a discrete linear quadratic controller at every time step.

Figures 6–8 describe the variations of the system variables in reaching the desired attitude. Figure 9 shows the spacecraft Euler angles with and without the hybrid nonlinear–linear control. We see that the nonlinear optimal controller, along with the linear controller, gets the spacecraft to the desired attitude from large angular deviations. Figure 10 shows the control (voltage) input to the three reaction wheels for the hybrid design. The control design phases from the nonlinear controller to the linear controller when $q_{1e}(k)>0.9997$ (angular error $<3^\circ$) and $\|\omega\|<0.0075$ rad/s and $\|\omega_{\text{RW}}\|<10$ rad/s. The linear controller is phased in gradually to minimize transient effects,

$$u(k)=\alpha(k)u_{\text{lin}}(k)+[1-\alpha(k)]u_{\text{nonlin}}(k) \tag{24}$$

where $\alpha(k)$ varies smoothly from 0 to 1 such as the sigmoidal function.

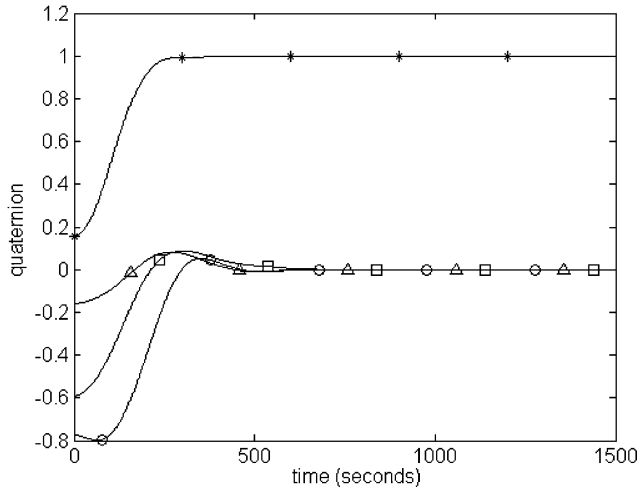


Fig. 6 Quaternion variations for the spacecraft: *, q_1; O, q_2; Δ, q_3; and □, q_4.

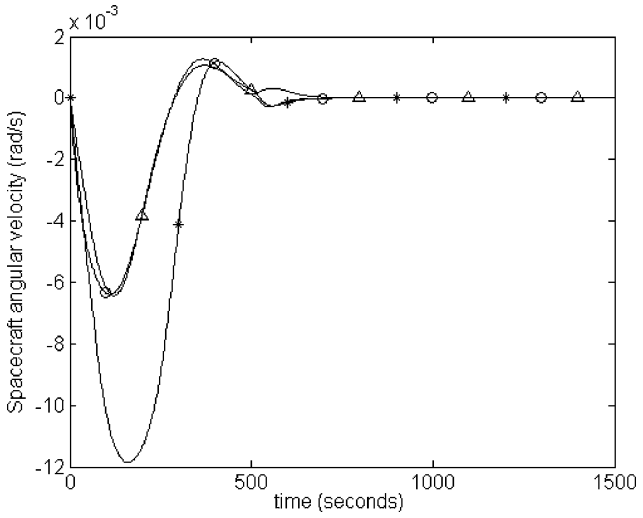


Fig. 7 Spacecraft angular velocities: *, ω_1; O, ω_2; and Δ, ω_3.

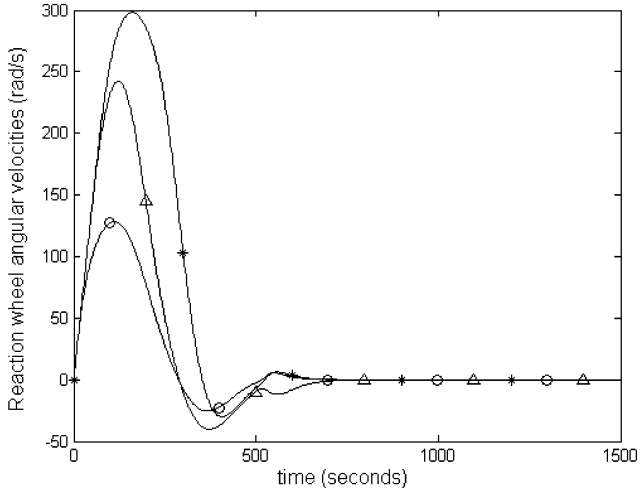
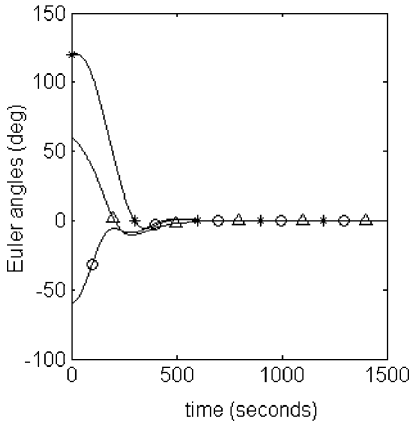
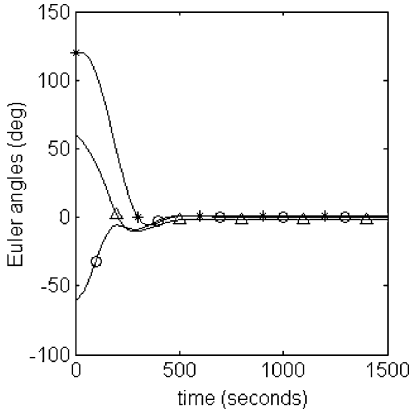


Fig. 8 Spacecraft reaction wheel angular velocities: *, ω_{RW1}; O, ω_{RW2}; and Δ, ω_{RW3}.



a)



b)

Fig. 9 Variation of the spacecraft Euler angles a) with the hybrid controller and b) with the nonlinear controller alone: *, θ_1; O, θ_2; and Δ, θ_3.

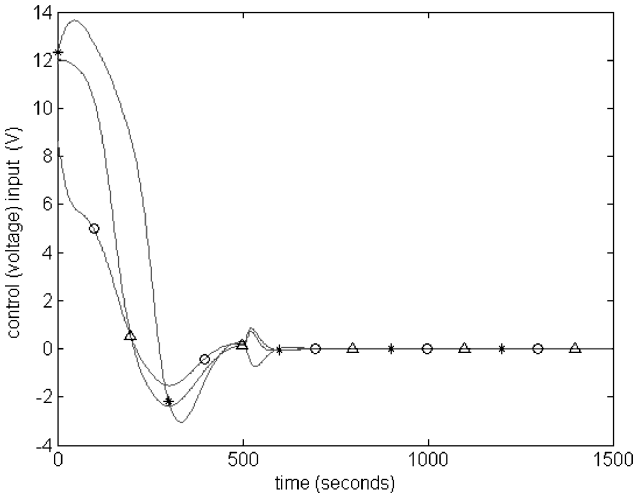


Fig. 10 Control input to the reaction wheels: *, v_1; O, v_2; and Δ, v_3.

VI. Conclusions

In this paper we have presented a general design method for the optimal control of nonlinear systems using neural networks. Either simulation data or actual experimental data can be used to design the controller without requiring an explicit model of the system in standard form. The training of the cost-to-go function is instrumental in the proposed controller-design process. A specific neural network architecture has been developed to approximate the cost-to-go function. The architecture allows systematic component-by-component construction of the cost-to-go network. Each building block of the

cost-to-go network can be trained invidually without regard to the others. This decentralized strategy eliminates many uncertainties associated with using artificial neural networks in this type of application. Once the cost-to-go network is trained, the controller can be trained to be optimal without the cost-to-go network having to be retrained. This decoupling of the training of the two networks is an attractive feature of our approach. In our approach the cost-to-go network is a function not only of the present system state, but also of present and future control action. When coupled with an optimally trained controller, this combination network outputs the optimal cost-to-go value. Only then does the optimal cost to go become a function of the present system state alone, as expected in optimal control theory. Thus from the perspective of adaptive critic design, our entire network is a single critic.

The control strategy follows the same principles as nonlinear predictive control. The novel feature here lies in the use of multiple predictor subnets in building the cost-to-go network. Nonlinear-predictive-control architectures typically use a one-step system model to design the controller instead of blocks of single- and multiple-step-ahead predictors as used here. Using multiple-step-ahead predictors helps parallelize the architecture of the cost-to-go function approximator network, which reduces the error in the back-propagation derivatives needed to train the controller subnetworks. The controller design is shown to successfully orient a spacecraft using reaction wheels through large angular deviations.

Acknowledgments

This research is supported by the National Science Foundation through the ANSER Corporation and also by NASA. The authors acknowledge Princeton Satellite Systems, Princeton, New Jersey, for providing the spacecraft model.

References

[1]Werbos, P. J., "Approximate Dynamic Programming for Real Time Control and Neuro-Modeling," *Handbook of Intelligent Control*, edited by D. A. White and D. A. Sofge, Van Nostrand Reinhold, New York, 1992, pp. 493–526.

[2]Balakrishnan, S. N., and Biega, V., "Adaptive Critic Based Neural Networks for Aircraft Optimal Control," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 4, 1996, pp. 893–898.

[3]Prokhorov, D., and Wunsch, D. C., "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997–1007.

[4]Ferrari, S., and Stengel, R. F., "An Adaptive Critic Global Controller," *Proceedings of the American Control Conference*, IEEE Press, Piscataway, NJ, 2002, pp. 2665–2670.

[5]Kulkarni, N. V., and Phan, M. Q., "Data-Based Cost-to-Go Design for Optimal Control," AIAA Paper 2002-4668, Aug. 2002.

[6]Cybenko, G., "Approximation by Superposition of Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, Vol. 2, No. 4, 1989, pp. 303–314.

[7]Hornik, K., Stinchcombe, M., and White, H., "Multilayer Feedforward Networks as Universal Approximators," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 331–409.

[8]Werbos, P. J., McAvoy, T., and Su, T., "Neural Networks, System Identification, and Control in Chemical Process Industries," *Handbook of Intelligent Control*, edited by D. A. White and D. A. Sofge, Van Nostrand Reinhold, New York, 1992, pp. 283–356.

[9]Goodwin, C. G., and Sin, K. S., *Adaptive Filtering, Prediction, and Control*, Prentice–Hall, Englewood Cliffs, New Jersey, 1984.

[10]Narendra, K. S., and Annaswamy, A. M., *Stable Adaptive Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1989.

[11]Mosca, E., *Optimal, Predictive and Adaptive Control*, Prentice–Hall, Englewood Cliffs, NJ, 1995.

[12]Norgaard, M., Ravn, O., Poulsen, N. K., and Hansen, L. K., *Neural Networks for Modeling and Control of Dynamic Systems*, Springer-Verlag, London, 2000.

[13]*Spacecraft Control Toolbox*, Ver. 5.5, Princeton Satellite Systems, Princeton, NJ.

[14]Hagan, M. T., and Menhaj, M. B., "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, 1994, pp. 989–993.

[15]Reidmiller, M., and Braun, H., "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, 1993, pp. 586–591.

[16]Wie, B., *Space Vehicle Dynamics and Control*, AIAA Education Series, AIAA, Reston, VA, 1998, pp. 402–405.

[17]Kulkarni, N. V., and Phan, M. Q., "A Neural Networks Based Design of Optimal Controllers for Nonlinear Systems," AIAA Paper 2002-4664, Aug. 2002.