Neural Dynamic Optimization for Control Systems—Part III: Applications

Chang-Yun Seong, *Member, IEEE,* and Bernard Widrow, *Life Fellow, IEEE*

***Abstract*—The paper presents neural dynamic optimization (NDO) as a method of optimal feedback control for nonlinear multi-input-multi-output (MIMO) systems. The main feature of NDO is that it enables neural networks to approximate the optimal feedback solution whose existence dynamic programming (DP) justifies, thereby reducing the complexities of computation and storage problems of the classical methods such as DP. This paper demonstrates NDO with several applications including control of autonomous vehicles and of a robot-arm, while the two other companion papers of this topic describes the background for the development of NDO and present the theory of the method, respectively.**

***Index Terms*—Autonomous vehicles, dynamic programming, information time shift operator, learning operator, neural dynamic optimization, neural networks, nonlinear systems, optimal feedback control, robots.**

I. Introduction

In this paper, we apply neural dynamic optimization (NDO) to several control problems: the lateral autopilot logic for a Boeing 747, the minimum fuel control of a double integrator plant (DIP) (e.g., a satellite) with bounded control effort, the automatic backward steering of a truck with two trailers, and the set-point control of a two-link robot-arm. The lateral autopilot logic for a Boeing 747 demonstrates that NDO closely approximates the optimal solution to a linear quadratic problem that dynamic programming (DP) [i.e., linear quadratic regulator (LQR)] would produce. The minimum fuel control of a DIP illustrates that NDO enables neural networks to closely approximate the optimal feedback solution of a nonlinear control problem. The backward steering of a truck with two trailers illustrates that NDO can produce a larger domain of attraction for regulation problems than LQR because it does not suffer from the limitations of linearization. Finally, the set-point control of a two-link robot-arm demonstrates that NDO works for a nonlinear multi-input-multi-output (MIMO) control problem.

The background for the development of NDO and the theory of NDO are presented in the two other companion papers [1], [2], respectively.

II. Boeing 747 Airplane

The Boeing 747 airplane is one of the most capable transports ever built. Because of its extensive range (resulting in pilot fatigue), and a desire to minimize the crew requirements, a capable controller (e.g., autopilot) is required in the aircraft design. Along with this motivation, our autopilot design process demonstrates that NDO closely approximates the optimal solution to a linear quadratic problem that DP (i.e., LQR) would produce.

Let us consider a linear-time-invariant MIMO model of the Boeing 747 lateral motion obtained from [3]. The linear equations of the lateral motion are compactly represented in state-space form. The four states and two inputs as defined as follows:

$$x(t) = \begin{bmatrix} \text{Sideslip angle, } \beta(t), \text{ in degrees} \\ \text{Yaw rate, } r(t), \text{ in } \frac{\text{degrees}}{\text{second}} \\ \text{Roll rate, } p(t), \text{ in } \frac{\text{degrees}}{\text{second}} \\ \text{Bank angle, } \phi(t), \text{ in degrees} \end{bmatrix}$$

and

$$u(t) = \begin{bmatrix} \text{Rudder angle, } \delta r(t), \text{ in degrees} \\ \text{Aileron angle, } \delta a(t), \text{ in degrees} \end{bmatrix}.$$

Then,

$$\dot{x}(t) = Ax(t) + Bu(t)$$

where

$$A = \begin{bmatrix} -0.0558 & -1.0000 & 0.0000 & 0.0416 \\ 0.5983 & -0.1150 & -0.0318 & 0.0000 \\ -3.0496 & 0.3880 & -0.4650 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0073 & 0.0000 \\ -0.4750 & 0.0077 \\ 0.1530 & 0.1430 \\ 0.0000 & 0.0000 \end{bmatrix}.$$

The lateral motion of the airplane is *unstable* because its system matrix A has two stable real eigenvalues, but a pair of unstable complex eigenvalues

$$\lambda_1 = -0.0099$$
$$\lambda_2 = -0.6837$$
$$\lambda_{3,4} = 0.0289 \pm 0.8538j.$$

Manuscript received January 16, 2000; revised February 11, 2001. This work was supported in part by the National Aeronautics and Space Administration under Contract NCC2-1020; the Office of Naval Research under Contract N00014-97-1-0947; and the National Science Foundation under Contract ECS–9522085. This paper was recommended by Associate Editor L. O. Hall.

C. Y. Seong was with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA. He is now with Fineground Networks, Campbell, CA 95008 USA (e-mail: chang@fineground.com).

B. Widrow is with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: widrow@stanford.edu)

Publisher Item Identifier S 1083-4419(01)05667-9.

Our goal is to compute a state feedback law to improve and stabilize this lateral motion. We check the controllability of the linear system so that we can guarantee the existence of a stabilizing feedback control. This system turns out to be controllable.

We discretize these linear system equations from continuous-time to discrete-time using the command C2DM in MATLAB. Here are the discrete LTI-MIMO system equations of the Boeing 747 lateral motion (for $T = 0.2$ s)

$$x[k+1] = A_d x[k] + B_d u[k]$$

where

$$A_d = \begin{bmatrix} 0.9769 & -0.1958 & 0.0014 & 0.0082 \\ 0.1190 & 0.9652 & -0.0059 & 0.0005 \\ -0.5722 & 0.1313 & 0.9107 & -0.0024 \\ -0.0585 & 0.0114 & 0.1910 & 0.9998 \end{bmatrix}$$

$$B_d = \begin{bmatrix} 0.0108 & -0.0001 \\ -0.0935 & 0.0014 \\ 0.0234 & 0.0274 \\ 0.0026 & 0.0028 \end{bmatrix}.$$

Now we want to find the optimal weight matrix $\mathbf{W}$ of the linear neural network $u[k] = \mathbf{W}x[k]$, minimizing a constant-coefficient quadratic cost function

$$J = \frac{1}{2}x^{\mathrm{T}}[N]Qx[N] + \frac{1}{2}\sum_{k=0}^{N-1}\{x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k]\} \quad (1)$$

where the weight matrix $\mathbf{W}$ is a 2×4 matrix, the matrix Q is a symmetric positive semidefinite 4×4 matrix, and the matrix R is a symmetric positive definite 2×2 matrix. The state weighting matrix Q and control weighting matrix R provide some compromise between the speed of response and the use of control effort. As we increase the eigenvalues of the matrix Q with respect to those of the matrix R, the system response gets faster. In this case, we pick $Q = I_{4\times4}$ and $R = 0.1\,I_{2\times2}$. In addition, we want to select a sufficiently large time horizon N so that the solution to this finite-horizon problem can converge to the solution to the corresponding infinite horizon problem. Therefore, we choose $N = 40$ as the time horizon.

Table I shows the NDO result. For comparison, we also show the LQR result obtained by the command DLQR in MATLAB with the same weighting matrices, Q and R. Both results are identical up to the fourth decimal digit. NDO approximates very closely the feedback solution that the LQR (derived from DP) produces. In addition, Fig. 1 plots the result of FOC along with those of both NDO and DP (LQR). We apply FOC to this optimal control problem using the same time horizon N as well as the same weighting matrices, Q and R. All the state trajectories and control profiles are identical, as illustrated in Fig. 1.

This example demonstrates that NDO closely approximates the optimal solution that both DP (LQR) and FOC produce although NDO takes a different approach.

III. Minimum Fuel Control of a Double Integrator Plant with Bounded Control Effort

This section considers a terminal control problem, where the control effort (e.g., fuel or energy) required is the criterion of optimality. Such problems arise frequently in aerospace applications, where there are limited control resources available for achieving desired objectives. Suppose we wish to control a DIP like a satellite

$$\ddot{p} = u(t) \quad (2)$$

where p is the position of the plant. In addition, its maximum control input is limited as a physical constraint

$$|u(t)| < u_0.$$

The original system is linear, as shown in (2). However, the overall system equation is nonlinear because of the physical constraint.

Our control objective is to find the paths from states in the $(p, \dot{p})$ space to $p = \dot{p} = 0$ in a specified final time t_f, which minimize fuel, i.e., one-norm of $u(t)$

$$J = \int_0^{t_f} |u(t)|\, dt. \quad (3)$$

This optimal control problem is selected from [4]. This example demonstrates that NDO enables neural networks to closely ap-

TABLE I
Comparison of NDO and LQR Results

• NDO result:

$$\mathrm{W}_{\mathrm{NDO}} = \begin{bmatrix} -7.7975 & 6.7786 & 2.3790 & 1.3693 \\ 4.1822 & -1.9604 & -2.4096 & -1.7750 \end{bmatrix}$$

• LQR result:

$$\mathrm{W}_{\mathrm{LQR}} = \begin{bmatrix} -7.7975 & 6.7786 & 2.3790 & 1.3693 \\ 4.1822 & -1.9604 & -2.4096 & -1.7750 \end{bmatrix}$$

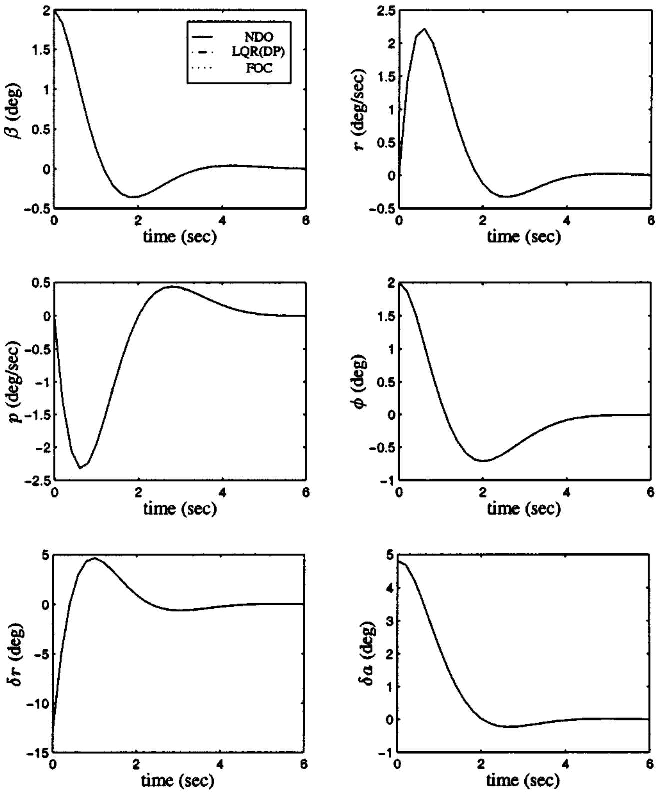

Fig. 1. Comparison of the results of NDO, DP, and FOC.

proximate the same optimal solution to nonlinear control problems that both DP and FOC find. It also illustrates that NDO produces the controllers that are able to reject input disturbances.

Let us discretize the DIP equation in order to apply NDO. Here is the zero-order-hold (ZOH) equivalent of the (2) in the state space form

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix} + \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix} u[k] \tag{4}$$

where $x_1 = p$ and $x_2 = \dot{p}$.

Then we want to find the optimal weight vector W of the multilayer feedforward sigmoidal neural network $u[k] = g(x[k], k; W)$, minimizing

$$J = \sum_{k=0}^{N-1} |u[k]| + x^{\mathrm{T}}[N] S_f x[N] \tag{5}$$

subject to

$$|u[k]| < u_0 \tag{6}$$

where $N = t_f/T_s$ and S_f is a weighting matrix of the final state. In this case, we pick the sampling time $T_s = 0.02\,t_f$, which results in $N = 50$. Along with fuel usage, we penalize the error of the final state with the matrix S_f because we are also concerned about it. Note that we slightly modify the cost function (3) by adding the soft terminal constraint term

$$x^{\mathrm{T}}[N] S_f x[N]. \tag{7}$$

Without this term, the cost function would be a cost function with a hard terminal constraint. If we pick a large enough weighting matrix of the soft terminal constraint S_f, the cost function with soft terminal constraint will approximate the cost function with hard terminal constraint because a fraction of the cost of the soft terminal constraint is negligible compared to a fraction of the cost of fuel. Therefore, we select $S_f = \mathrm{diag}\{10^5, 10^5\}$. Note that we allow the neural network to have explicit time-dependency so that it can approximate optimal time-varying feedback solutions although the system is nonlinear-time-invariant (NTI) and the cost function has constant coefficients. In this application, the explicit time-dependency is necessary for the neural network to approximate the optimal solution closely over a range of state space. We discuss the matter in detail at the end of this section.

Let us consider the following range of initial states, for the neural controller to optimize the performance of the closed-loop system over a family of trajectories associated with different values of initial states

$$\{0.1 < |x_1[0]| < 0.4, |x_2[0]| < 0.05\} \tag{8}$$

where x_1 and x_2 are in units of $u_0 t_f^2/2$ and $u_0 t_f$, respectively. We want to drive the system from any initial state in the range to the origin at time t_f, consuming the minimum amount of fuel. The neural network used in the controller consists of a two-layer feedforward sigmoidal network with three inputs, zero hidden units, and one output (denoted as a $\mathcal{N}_{3:10:1}$). Note that the neural network has three inputs: because of its explicit time dependency, it receives time k from the clock as well as the current states $x_1[k]$ and $x_2[k]$. We scale the value of time k by 0 so that it falls roughly between 0 and +1 before being fed into the neural network. We also scale the value of $x_1[k]$ by $0.2\,u_0 t_f^2$ while we

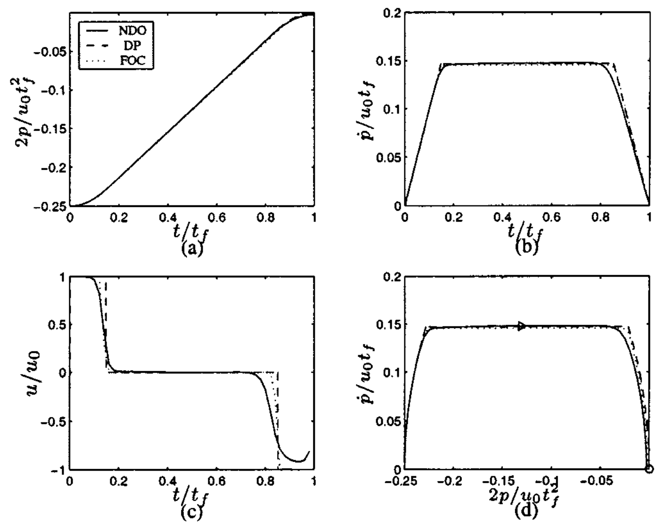

Fig. 2. NDO result for the range of the training set of initial states $\{(x_1, x_2) \mid 0.1 < |x_1| < 0.4, |x_2| < 0.05\}$. (a) Position. (b) Velocity.(c) Control force. (d) Phase plane plot. The symbols ○ and —▷ represent the destination and the directions of the trajectories, respectively.

do not scale $x_2[k]$. Thus including the bias input weights, the total number of its weights is $n_w = (3+1) \times 10 + 10 \times 1 = 50$, which is equal to the multiplication of the number of time steps and the number of control inputs (FOC also requires the same amount of storage locations to store the optimal control sequence, $u[k],\ k = 0, \ldots, 49$). Recall Rule 1 for selecting the number of weights discussed in Section IV-D of the counterpart of this paper. It recommends that NDO should employ at least a number of weights greater than the multiplication of the number of time steps and the number of control inputs, to guarantee full column rank of the learning operator and prevent a loss of information in updating the weight vector. Moreover, the neural controllers with sigmoidal nonlinearities such as $hbox\tanh(\cdot)$ at the output layer have a natural way to account for bounded control effort because their sigmoidal activations have built-in saturation limits between -1 and 1. Thus the bounded control constraint is readily handled by the neural controller using a proper scaling factor. We initialize the weights of the first layer of the network using Nguyen's method [5] while we set the initial weights of its second layer to uniformly distributed random values between -0.01 and 0.01. We pick a learning rate $\mu = 2 \times 10^{-4}$ by experiment. The total number of iterations is 2×10^7. The initial state $x[0]$ for each iteration is chosen uniformly over the given range of initial states. Along with the NDO results, we also present the results of both DP and FOC to compare them. The FOC results are obtained by using the discretized system (4) and the soft-terminal-constraint cost function (5) with the same control design parameters that NDO uses, while the DP solution is obtained by employing the continuous-time system (2) and the hard-terminal-constraint cost function (3).

The optimal solution to the minimum fuel control with bounded control effort has a very interesting characteristic: bang-off-bang, as illustrated in Fig. 2. The optimal control applies maximum acceleration in the beginning, and then coasts without any driving forces, and applies maximum deceleration

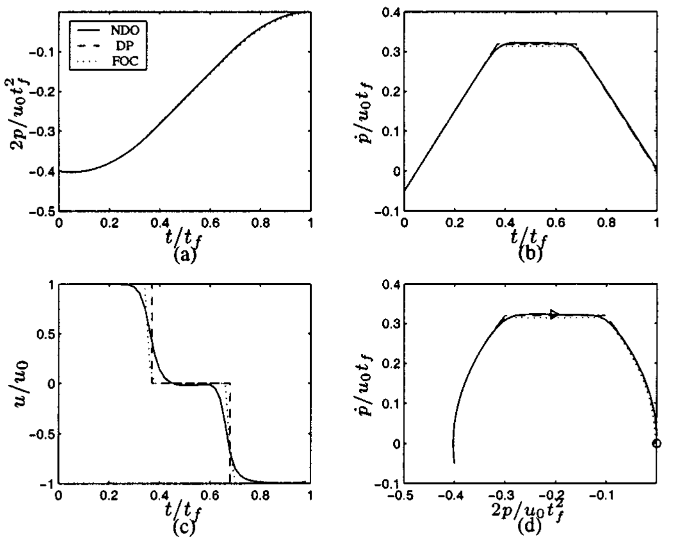

Fig. 3. NDO result for the range of the training set of initial states $\{(x_1, x_2) \mid 0.1 < |x_1| < 0.4, |x_2| < 0.05\}$. (a) Position. (b) Velocity. (c) Control force. (d) Phase plane plot. The symbols ∘ and ⊸▹ represent the destination and the directions of the trajectories, respectively.

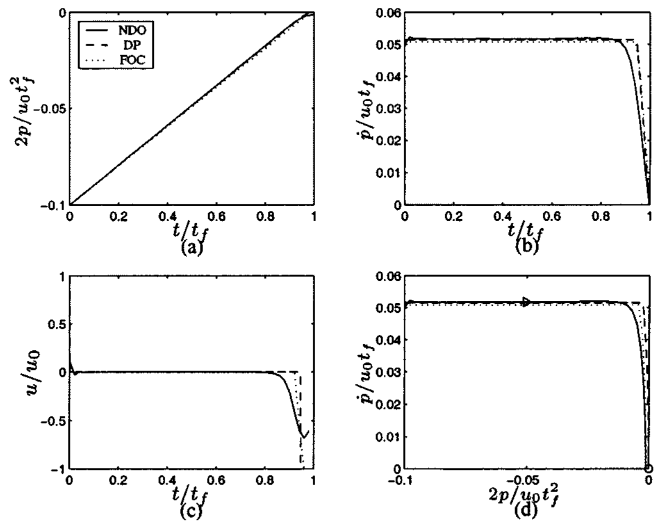

Fig. 4. NDO result for the range of the training set of initial states $\{(x_1, x_2) \mid 0.1 < |x_1| < 0.4, |x_2| < 0.05\}$. (a) Position. (b) Velocity. (c) Control force. (d) Phase plane plot. The symbols ∘ and ⊸▹ represent the destination and the directions of the trajectories, respectively.

in the final stage to minimize the fuel usage. As the reader can see, the results of NDO are close to those of both DP and FOC. We particularly point out that the neural controller captures the discontinuous characteristic—bang-off-bang—of the optimal solution.

Figs. 3 and 4 present the results of NDO for two extreme points of the training set: the former shows the case where $x_1[0] = -0.4$ and $x_2[0] = -0.05$; the latter shows the case where $x_1[0] = -0.1$ and $x_2[0] = 0.05$. The neural controller successfully drives the system from these initial states to the destination at $t = t_f$. In addition, the control profiles of the neural controller as well as the trajectories of the closed-loop system are close to the optimal solution of both DP and FOC.

What happens if the dynamical system confronts disturbances and uncertainties in real-world applications? Controlled sys-

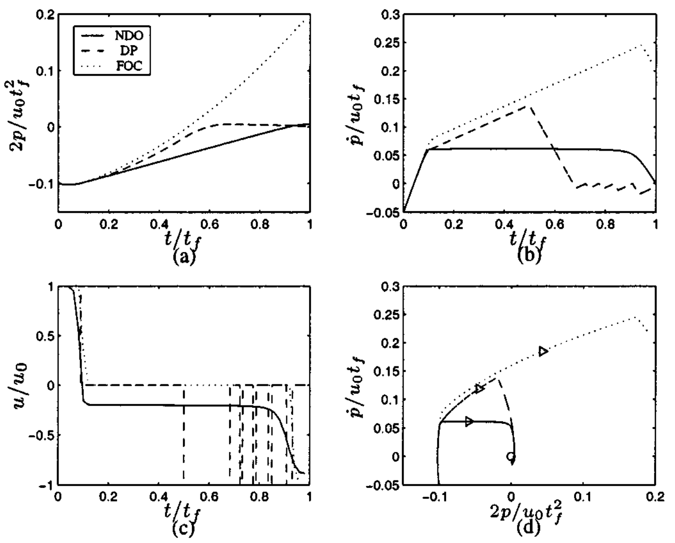

Fig. 5. Disturbance-rejection tests of the neural controller against constant input disturbances. (a) Position. (b) Velocity. (c) Control force. (d) Phase plane plot. The symbols ∘ and ⊸▹ represent the destination and the directions of the trajectories, respectively. Note that the neural controller cancels the effect of the input disturbance during its coasting by applying a counteracting force, thereby enabling it to bring the system closely to the destination.

tems are often designed to minimize their effects. In order to test the disturbance-rejection capabilities of NDO, we intentionally apply to the controlled system a constant input disturbance of $-0.2u_0$, whose magnitude is 0% of its maximum control force. Fig. 5 shows the disturbance-rejection test results of NDO, DP, and FOC. The neural controller and DP solution drive the system closely to the desired destination in time despite the input disturbance. In contrast, the FOC solution fails because it produces an open-loop solution. We point out that the neural controller cancels the effect of the input disturbance during its coasting by applying a counteracting force of $+0.2u_0$ so that it can bring the system to the desired destination in time. In other words, the neural controller takes into account the input disturbance and subtracts the effect to reach the destination because the neural controller responds to feedback or changes in the system state. The DP solution also takes advantage of the system-state information to switch its control input from one value to another, thereby driving the system closely to the destination. However, its switching mechanism may cause a chattering in the system, as illustrated in Fig. 5.

Fig. 6 shows the normalized phase-plane results for the minimum fuel control. The horizontal axis represents the normalized position and the vertical axis represents the normalized velocity. The dotted lines represent the trajectories of the closed-loop system whose controller NDO trained over the training set of initial states $\{0.1 < |x_1[0]| < 0.4, |x_2[0]| < 0.05\}$: they are accelerating in the beginning, coasting in the middle, and decelerating to arrive at the origin in the final stage. We also plot the results of the neural controller for some initial states outside the training set. Dash-dot lines represent constant contours of the normalized optimal cost function. The solid lines are the optimal trajectories DP produces. The dashed line is the optimal switching curve. Note that all the optimal trajectories come along with

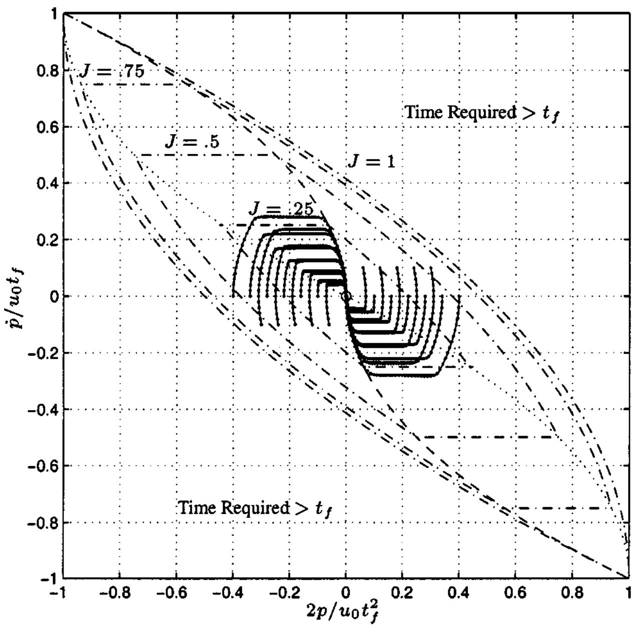

Fig. 6. Normalized phase-plane result of NDO for the minimum fuel path control with bounded control effort, where u_0 is a maximum driving force. The dotted lines represent the trajectories of the closed-loop system whose controller NDO trained. The solid lines represent the optimal trajectories DP produces. The dashed line is the optimal switching curve. The dash-dot lines are constant contours of the normalized optimal return function. Note that the neural controller captures not only discontinuous characteristics of the optimal control solution such as bang-off-bang but also the optimal switching curve.

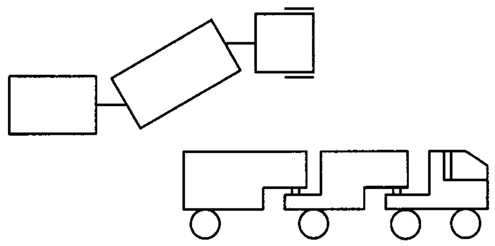

Fig. 7. Truck with two trailers (top and side views).

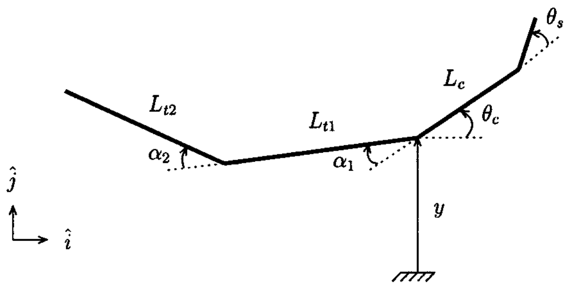

Fig. 8. Geometry of a truck with two trailers.

the optimal switching curve, while approaching the origin. Note also that the optimal control always applies maximum deceleration along the optimal switching curve whenever the optimal trajectories meet it and come along with it. These observations imply that the optimal feedback control must have explicit time-dependency in order to produce the family of the optimal trajectories over the range of state space. As the reader can see, the results of NDO are close to those of DP: the dotted lines are on top of the solid lines. In other words, NDO closely approximates the known optimal feedback solution. The resulting neural controller captures not only discontinuous characteristics such as bang-off-bang of the optimal control solution but also the optimal switching curve.

This example demonstrates that NDO enables neural networks to closely approximate the optimal feedback solution to nonlinear control problems. It also shows that neural controllers NDO produces are able to reject the input disturbances ubiquitous in applications.

IV. Backing a Truck with Two Trailers

This section considers the automatic backward steering of a truck (cab) with two trailers. Its top and side views are depicted in Fig. 7, where the truck and trailers are all connected to each other by joints. Our control goal is to back up the truck with two trailers to a line ($y = 0$) and keep it moving backward along the line. The controller is required to steer the truck back to the desired steady state in a timely and robust fashion. This is a regulation control, i.e., an infinite-horizon problem. The application will illustrate that NDO can produce a larger domain of attraction for regulation problems than LQR because NDO requires no linearization around an equilibrium point.

In fact, a different kind of truck backup problem has been investigated by Nguyen and Widrow [6]. Their goal is to back up a truck with one trailer to a loading dock at a final time. They care about a position error only at the final time where the desired state of the truck-trailer system is known. Once the end of its trailer reaches the loading dock, the control task is over. That is a terminal control problem. They successfully trained a neural network to achieve their goal using the backpropagation-through-time algorithm.

Our proposed problem is an infinite-horizon problem while Nguyen's problem is a finite-horizon problem. In addition, we add another trailer in order to increase the complexities of the truck-trailer system; backing up the truck with two trailers is a harder task than backing up the truck with one trailer. The geometry of the truck with two trailers appears in Fig. 8; the angle θ_s represents the steering angle, the angle θ_c represents the orientation of the cab with respect to an inertial frame, α_1 represents the joint angle between the cab and the first trailer, α_2 represents the joint angle between the first and the second trailer, L_c represents the length of the cab, L_{t1} represents the length of the first trailer, and L_{t2} represents the length of the second trailer.

Here are the system equations for the truck backup to a line ($y = 0$) with two trailers

$$\frac{dy}{ds} = -\sin\theta_c \tag{9}$$

$$\frac{d\theta_c}{ds} = -\frac{\tan\theta_s}{L_c} \tag{10}$$

$$\frac{d\alpha_1}{ds} = \frac{\sin\alpha_1}{L_{t1}} - \frac{\tan\theta_s}{L_c} \tag{11}$$

$$\frac{d\alpha_2}{ds} = \frac{\cos\alpha_1 \sin\alpha_2}{L_{t2}} - \frac{\sin\alpha_1}{L_{t1}} \tag{12}$$

where y represents j-component of the position of the joint between the cab and the first trailer in the inertial frame. Let us denote $x \triangleq [y\ \theta_c\ \alpha_1\ \alpha_2]^{\mathrm{T}}$, and $u \triangleq \theta_s$. In this application, we set $L_c = 0.625\,L_{t1}$ and $L_{t2} = L_t$. Note that the independent variable is not time but the distance traveled, thereby allowing a controller to back up the truck with *nonconstant* speed. That's why the speed of the truck does not appear in the equations of motion.

As physical constraints, joint angles, α_1 and α_2, cannot exceed 60°. The truck with two trailers cannot move backward anymore whenever either joint angle reaches 60°. We call this angle a jackknife. In addition, the maximum steering angle is 30°. The system equations are highly nonlinear because of jackknives, the saturation of the steering angle, and the nonlinear terms like $\cos\alpha_1 \sin\alpha_2$. In this application, particularly, we pretend that the mathematical model of the system does not exist (we can be confronted by such situations in practice) but the data of the system are available. The truck and trailer system can therefore be computer-simulated. We train another neural network, the so-called *system emulator*,[1] to model (or approximate) the simulated system using the data produced by numerical integration (Runge-Kutta method) of the differential (9)–(12) with integration interval $T_s = 0.1\,L_{t1}$. For each numerical integration, the next state is determined by the present state and the steering angle, which is fixed during the integration interval. The emulator has five inputs corresponding to the four state variables $x[k]$ and the steering angle $u[k]$, and four outputs corresponding to the four next state variables $x[k+1]$. The training set for the emulator is

$$\{|x_1| < 15\,L_{t1}, |x_2| < \pi \text{ rad}, |x_3| < \frac{\pi}{3} \text{ rad}, |x_4| < \frac{\pi}{3} \text{ rad}\}.$$

In particular, we scale the values of $x[k]$ and $u[k]$ so that they fall between $+1$ and -1 before being fed into the emulator. The neural network used in the emulator is $\mathcal{N}_{5:20:4}$. The architecture of this network is determined by experiment. The initial weights are set to uniformly distributed random values between -0.01 and 0.01.

During the emulator training, the truck backs up randomly, going through many cycles with randomly selected steering angles. The emulator learns to generate the next state when given the present state and steering angle, doing so for a wide variety of the states and steering angles. This neural emulator is trained by the backpropagation algorithm [7], [8]. By this process, the emulator eventually learns how the truck-trailer system behaves. Fig. 9 shows the learning curve of the emulator with a learning rate $\mu = 10^{-3}$, which indicates the reduction of square error during the iterative process. Once the emulator is trained, it can be used to train the neural controller.

Then, we want to find the optimal weight vector W of the multilayer feedforward sigmoidal neural network $u[k] = g(x[k]; W)$, minimizing a constant-coefficient quadratic cost function

$$J = \frac{1}{2}x[N]^{\mathrm{T}}Qx[N] + \frac{1}{2}\sum_{k=0}^{N-1}\{x[k]^{\mathrm{T}}Qx[k] + u^{\mathrm{T}}[k]Ru[k]\}$$

subject to

$$|x_{3,4}| < \frac{\pi}{3} \text{ rad}$$

[1]The process of training a system emulator corresponds to system identification in the control literature.

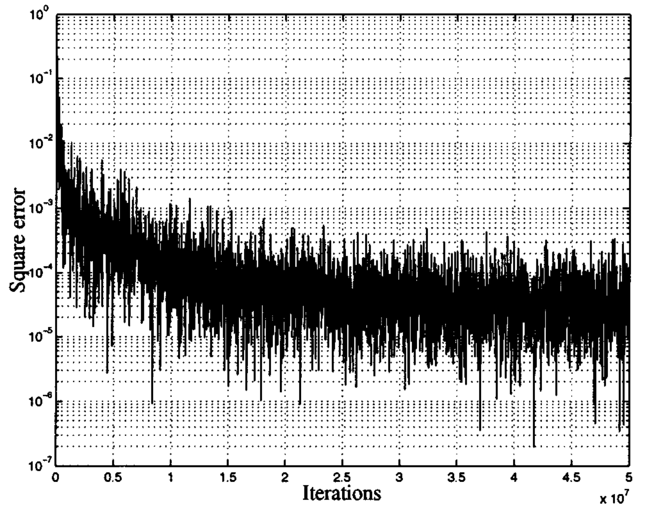

Fig. 9. Learning curve of the neural emulator: $\mathcal{N}_{5:20:4}$ is employed. Its learning rate is $\mu = 10^{-3}$.

and

$$|u| < \frac{\pi}{6} \text{ rad}$$

where the state weighting matrix Q is a symmetric positive semidefinite 4×4 matrix, and the input weighting matrix R is a symmetric positive definite 1×1 matrix. Note that we particularly employ the quadratic cost function because we want to compare the results of NDO and those of LQR while applying the same cost function, thereby allowing us to compare their performances. In this case, we pick $Q = \operatorname{diag}\{1, 0, 0, 1\}$, $R = 0.1$ as control design parameters. We penalize only the y-position of the truck and the second joint angle α_2 from $k = 1$ through N in order to achieve our goal, which is to back up the truck to the line $(y = 0)$ and keep it moving along the line. We select a sufficiently large time horizon $N = 240$ so that the solution to this problem can approximate the solution to the corresponding infinite-horizon problem. Note also that we do not include explicit time-dependency in the neural controller because we are working on the regulation problem involved with the NTI system and the constant-coefficient quadratic cost function.

The neural network employed in the controller is $\mathcal{N}_{4:10:1}$ with sigmoidal activation functions because we have four states and one control input of the controlled system. In particular, we set zero bias input to all the neurons of the neural network instead of $+1$. Therefore, the total number of the weights of the network is $n_w = 4\times 10 + 10\times 1 = 50$. Note that we employ a number of weights much smaller than Rule 1, discussed in Section IV-D of the counterpart of this paper, recommends. It suggests that the total number of weights should be at least zero because we have zero time steps and one control input. Instead of it, we follow Rule 2, which suggests that we need not increase the number of weights as the time horizon increases when NDO handles an infinite-horizon problem. It will be instructive to see how this neural controller performs, despite its possessing a number of weights even smaller than Rule 1 suggests. In addition, before feeding the state $x[k]$ into the controller, we scale the values of $x[k]$ by the same scaling factors that we employed in the training of the neural emulator. Moreover, the neural controllers with

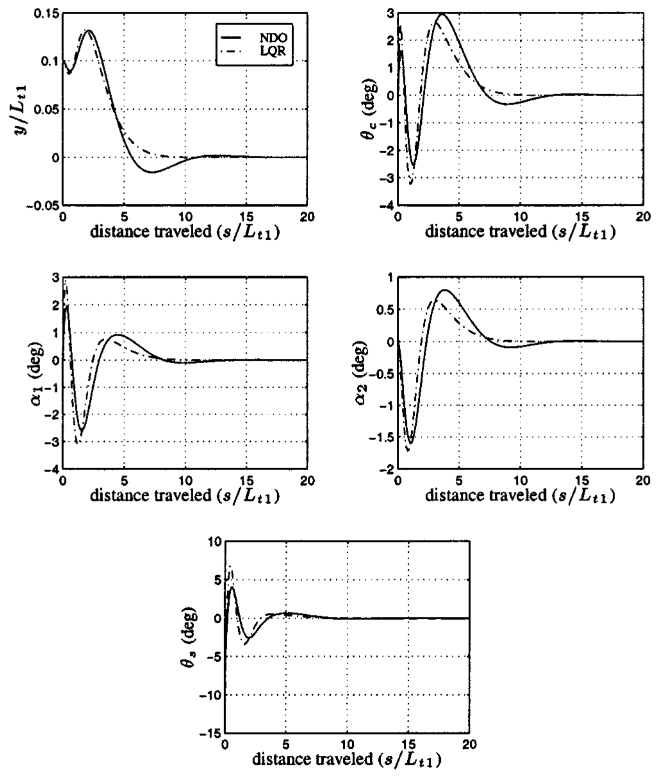

Fig. 10. NDO and LQR results for a small deviation from the equilibrium point, which is the origin. The LQR controller works better than the neural controller for small deviations from the equilibrium point.

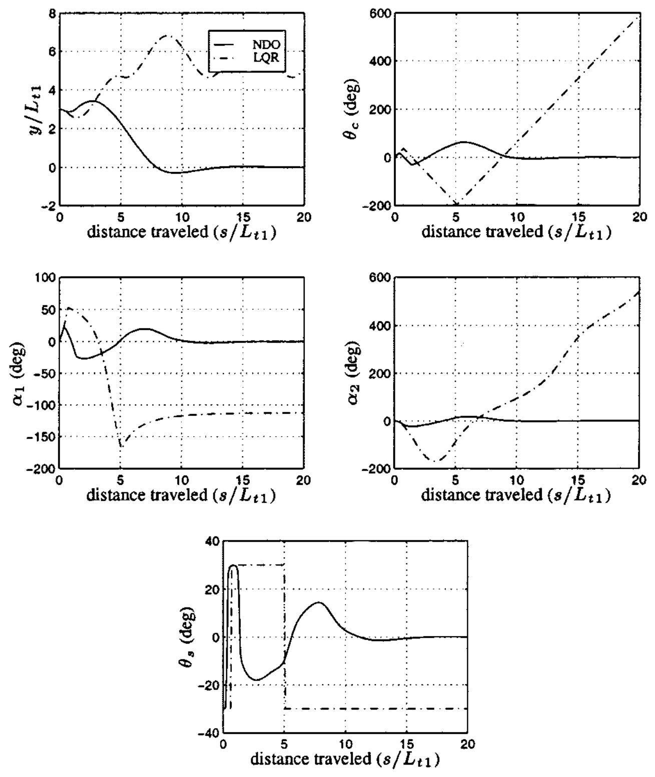

Fig. 11. NDO and LQR results for an intermidate deviation from the equilibrium point, which is the origin. As the deviations increase, the LQR controller fails because of the limitations of linearization. In contrast, the neural controller works because it doesn't suffer the limitations.

sigmoidal nonlinearities such as $\tanh(\cdot)$ at the output layer can readily handle bounded control efforts because the sigmoidal activations have built-in saturation limits between -1 and $+1$. Therefore, we can confine $u[k]$, the output of the neural controller, to the desired saturation limit by using the scaling factor $\pi/6$.

The initial state $x[0]$ for each iteration is drawn with a uniform probability distribution over the range of state space

$$\{|x_1| < 10\,L_{t1}, x_2 = 0\,\text{rad}, \text{x}_3 = 0\,\text{rad}, \text{x}_4 = 0\,\text{rad}\}.$$

We set the initial weights of the first layer of the neural controller using Nguyen's method [5] while we set those of the second layer to uniformly distributed random values between -0.01 and 0.01. The learning rate and the total number of iterations are 10^{-5} and 2×10^6, respectively. We train the neural controller using the *emulator* rather than the exact system equations.

In fact, Nguyen also used a neural emulator as part of training his neural controller. However, he used a system equation as well during his controller's training: he backs up the real truck (i.e., system equation) to get the *true* final state error, while he uses the neural emulator to adjust the weights of the neural controller using the error. Note that during its training his neural controller receives the *true* states of the system.

In contrast, our neural controller's training in this example completely *excludes* use of the real truck or system equation: we not only run our neural emulator to feed its states to the neural controller and evaluate the cost function but also use the emulator to adjust the controller's weights. Therefore, during its training our neural controller does not receive the true states of the system but the *approximated* states because the emulator is an approximated model of the system. The evaluation of the cost function is also *approximated* but not accurate. Once we finish training the neural controller by using the emulator, we obtain the NDO results by applying the trained controller to the real truck (i.e., system equation). Therefore, our trained neural controller must overcome *model uncertainties* resulting from the discrepancy between the true system and the neural emulator.

Along with the NDO results, we show the LQR results for comparison. When applying LQR, we use a *linear* model obtained by linearizing and discretizing the nonlinear system (9)–(12) around an equilibrium, which is the origin, $[y\ \theta_c\ \alpha_1\ \alpha_2]^{\mathrm{T}} = 0$. We obtain the LQR results by using the command DLQR in MATLAB with the same weighting matrices Q and R. The NDO and LQR results for small deviations from the equilibrium point appear in Fig. 10. For small deviations, the LQR controller works better than the neural controller. However, as the deviations increase the LQR controller fails because of the limitations of linearization, as illustrated in Fig. 11. In contrast, the neural controller works because it does not suffer the limitations.

Fig. 12 shows the NDO result for the largest vertical deviation among the initial states in the the training set. Initially, the truck with two trailers is set at the vertical distance $y = 10\,L_{t1}$ and parallel to the line $y = 0$. The neural controller successfully backs up the truck to the line and keeps it moving backward along the line. The neural controller works well for the large deviation from the equilibrium point.

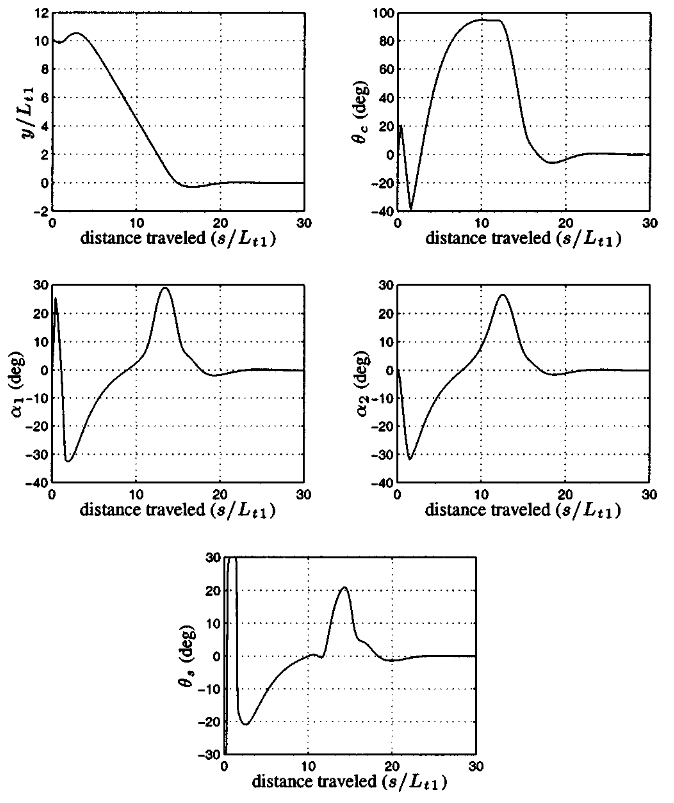

Fig. 12. NDO result for an large deviation from the equilibrium point, which is the origin. The neural controller successfully backs up the truck to the line and keeps it moving backward along the line.

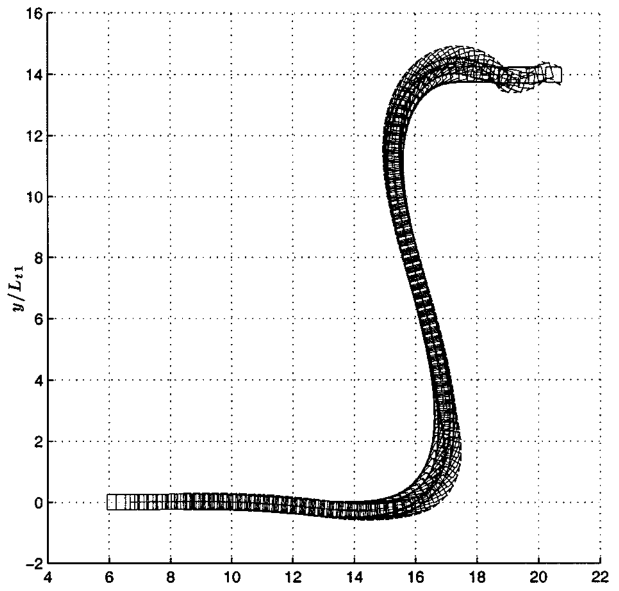

Fig. 13. Demonstration of the truck with two trailers backing up to the line $y = 0$. It is curious to see how the neural controller backs up the truck to the line. In the beginning it makes a sharp turn and then drives straight down and makes another sharp turn to the line $y = 0$, thereby backing up the truck to the line as soon as possible. The trajectory looks like a cobra—not the trajectory that common sense would have predicted.

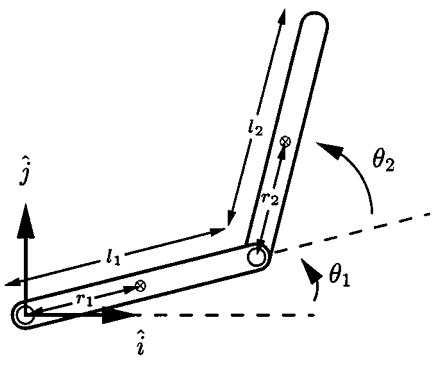

Fig. 14. Two-link planar robot-arm manipulator.

The demonstration of the truck backing up to the line $y = 0$ may reveal an aspect of optimal control. Suppose that the truck with two trailers is initially parallel to the line and located at $y = 14L_{t1}$, which is 40% larger than the largest vertical deviation among the initial states in the training set. Note that the line $y = 0$ is the position of the truck in steady state. It is interesting to see how NDO backs up the truck to the line in Fig. 13. In the beginning it makes a sharp turn and then drives straight down and makes another sharp turn to the line $y = 0$, thereby backing up the truck to the line as soon as possible. It is amazing that the turning angle is bigger than 90°. The trajectory looks like a cobra—not the trajectory that common sense would have predicted.

The example illustrates that NDO can produce a larger domain of attraction for regulation problems than LQR because NDO does not require linearization. It also illustrates that NDO is applicable by using a neural emulator (an approximated system model) if the mathematical model of a controlled system is not available. It shows that the neural controllers NDO produces can overcome model uncertainties resulting from the discrepancy between the true system and the neural emulator.

V. Two-Link Robot-Arm Manipulator

As a nonlinear MIMO control problem, this section considers a two-link planar robot-arm manipulator depicted in Fig. 14. Robot manipulators are familiar examples of trajectory-controllable mechanical systems. However, their nonlinear dynamics and highly coupled multiple inputs present a challenging control problem, since traditional linear control approaches do not easily apply.

For this manipulator, we define the angle of the first link θ_1 with respect to an inertial frame, as depicted in Fig. 14. We also define the angle of the second link θ_2 with respect to the orientation of the first link. We apply the torques τ_1 and τ_2 to control the angles θ_1 and θ_2, respectively. Here are the system equations for this manipulator [9], [10]

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}\alpha+2\beta c_2 & \delta+\beta c_2\\ \delta+\beta c_2 & \delta\end{bmatrix}\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} + \begin{bmatrix}-\beta s_2\dot{\theta}_2 & -\beta s_2(\dot{\theta}_1+\dot{\theta}_2)\\ \beta s_2\dot{\theta}_1 & 0\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}$$

where

$$c_2 = \cos(\theta_2),\ s_2 = \sin(\theta_2)$$

and

$$\begin{aligned}\alpha &= I_{z1} + I_{z2} + m_1 r_1^2 + m_2(l_1^2 + r_2^2)\\ \beta &= m_2 l_1 r_2\\ \delta &= I_{z2} + m_2 r_2^2.\end{aligned}$$

$I_{z(\cdot)}$ is the z-component of the inertial tensor of the $(\cdot)$th link around its center of mass, $m_{(\cdot)}$ is the link mass, $r_{(\cdot)}$ is the distance from the joint to the center of mass of the link, and $l_{(\cdot)}$ is the length of the link.

The values of the parameters are as follows:

$$\begin{aligned}m_1 &= 1 \text{ kg}\\ m_2 &= 2 \text{ kg}\\ l_1 &= 1 \text{ m}\\ l_2 &= 1.2 \text{ m}\\ r_1 &= 0.5 \text{ m}\\ r_2 &= \frac{l_2}{2} + \frac{0.5 m_3 l_2}{m_2 + m_3} \text{ m}\\ I_{z1} &= 0.12 \text{ kg}\cdot\text{m}^2\\ I_{z2} &= \frac{m_2 l_2^2}{12} + m_2\left(r_2 - \frac{l_2}{2}\right)^2 + m_3(r_2 - l_2)^2 \text{ kg}\cdot\text{m}^2.\end{aligned}$$

The third mass m_3 is a point-mass load at the end of the second link. It is considered to have a value of 0 kg under normal circumstances.

This manipulator is a highly nonlinear MIMO system, whose system equations include all highly nonlinear joint coupling terms (Coriolis and centripetal forces, variable effective moments of inertia, etc.). In addition, there exist the physical constraints for control inputs

$$\begin{aligned}|\tau_1(t)| &< 15 \text{ Nm}\\ |\tau_2(t)| &< 10 \text{ Nm}.\end{aligned}$$

Therefore, in order to achieve high performance such as high accuracy and speed, we cannot ignore the nonlinear forces through the linearization. We must take into account these nonlinear forces and deal with them properly.

We may apply NDO in order to fully compensate for the nonlinear dynamics as well as the physical constraints. First of all, we discretize the system equations by the Euler difference method using a sampling time $T_s = 0.05$ s. Then we want to find the optimal weight vector W of the multilayer feedforward sigmoidal neural network $u[k] = g(x[k], r[k]; W)$ minimizing

$$\begin{aligned}J =& \frac{1}{2}(x[N] - x_d)^{\mathrm{T}} Q (x[N] - x_d)\\ &+ \frac{1}{2}\sum_{k=0}^{N-1}\{(x[k] - x_d)^{\mathrm{T}} Q (x[k] - x_d)^{\mathrm{T}} + u^{\mathrm{T}}[k] R u[k]\}\end{aligned}$$

subject to

$$\begin{aligned}|\tau_1(t)| &< 15 \text{ Nm}\\ |\tau_2(t)| &< 10 \text{ Nm}\end{aligned}$$

where $x = [\theta_1\ \dot{\theta}_1\ \theta_2\ \dot{\theta}_2]^{\mathrm{T}}$, $u = [\tau_1\ \tau_2]^{\mathrm{T}}$, $r = [\theta_{1d}\ \theta_{2d}]^{\mathrm{T}}$, and $x_d = [\theta_{1d}\ 0\ \theta_{2d}\ 0]^{\mathrm{T}}$. In this case, we select $Q = \text{diag}\{1, 0.1, 1, 0.1\}$ and $R = 0.005 I_{2\times 2}$. Note that we penalize the joint angles, θ_1 and θ_2, much more strongly than the control inputs, τ_1 and τ_2, to achieve high speed and accuracy. In particular, we exclude explicit time-dependency in the neural controller. We choose the final time horizon $N = 140$.

The neural network used in the controller is $\mathcal{N}_{6:15:2}$ with sigmoidal activation functions. Including the bias input weights, the total number of its weights is $n_w = (6+1)\times 15 + 15\times 2 = 135$. Note that we employ a number of weights much smaller than Rule 1, discussed in Section IV-D of the counterpart of this paper, recommends. It suggests that the total number of weights should be at least zero because we have zero time steps and two control inputs. Instead, we follow Rule 2, which suggests that we need not increase the number of weights as the time horizon N increases when NDO handles an infinite-horizon problem. Moreover, the neural networks (or neural controllers) with the sigmoidal nonlinearities such as $\tanh(\cdot)$ at the output layer have a natural way to account for bounded control effort because the sigmoidal activation functions have built-in saturation limits between -1 and 1. Therefore, we can readily confine τ_1 and τ_2, the outputs of the neural controller, to the desired saturation limits by using proper scaling factors. However, we feed the state $x[k]$ into the controller without scaling the values of $x[k]$.

The initial state $x[0]$ and the desired steady state x_d for each iteration are chosen independently and uniformly over the range of the state space

$$\{|\theta_1| < 120, \dot{\theta}_1 = 0, |\theta_2| < 120, \dot{\theta}_2 = 0\}$$

where the angles θ_1, θ_2 are in units of degrees, and the angular rates $\dot{\theta}_1$, $\dot{\theta}_2$ are in units of degrees/seconds. The initial weights of the neural controller are set to uniformly distributed random values between -0.01 and 0.01. We pick a learning rate $\mu = 5\times 10^{-8}$. The total number of iterations is 2×10^6.

We present three groups of the NDO results for this manipulator. The first group shows how the neural controller works for the initial states inside the training set (note that all the initial velocities in the training set are zeros). The second group illustrates how the neural controller handles the initial states outside the training set. The third group tests the robustness of the controller by applying it to another two-link robot-arm possessing different system parameters.

Figs. 15–17 show some results of NDO for the initial states inside the training set. In particular, Figs. 15 and 16 demonstrate some important features of the results. Fig. 15 shows a full swing of the robot-arm over the trained state space. We set the initial state to $\theta_1[0] = -120\ ^\circ$, $\dot{\theta}_1[0] = 0\ ^\circ$/s, $\theta_2[0] = -120\ ^\circ$, and $\theta_2[0] = 0^\circ$/s. Then the neural controller drives the state to $\theta_1 = 120\ ^\circ$, $\dot{\theta}_1[0] = 0\ ^\circ$/s, $\theta_2 = 120\ ^\circ$, and $\dot{\theta}_2 = 0\ ^\circ$/s. NDO works well for those big rotations that the linear methods would be very difficult to solve for. In contrast, Fig. 16 shows how NDO handles directionality and dynamic coupling of a nonlinear MIMO system like the two-link robot-arm. For MIMO systems, especially, directionality is important because control inputs are coupled to each other through the system dynamics. In order to check how the neural controller deals with directionality, we apply the out-of-phase motions to the manipulator. We initially set the state to $\theta_1[0] = 120\ ^\circ$, $\dot{\theta}_1[0] = 0\ ^\circ$/s, $\theta_2[0] = -120\ ^\circ$, and $\theta_2[0] = 0\ ^\circ$/s. Then we drive the state to $\theta_1 = -120\ ^\circ$, $\dot{\theta}_1[0] = 0\ ^\circ$/s, $\theta_2 = 120\ ^\circ$, and $\theta_2 = 0\ ^\circ$/s. As Fig. 16 illustrates, NDO handles well directionality and two-way interactions of the manipulator.

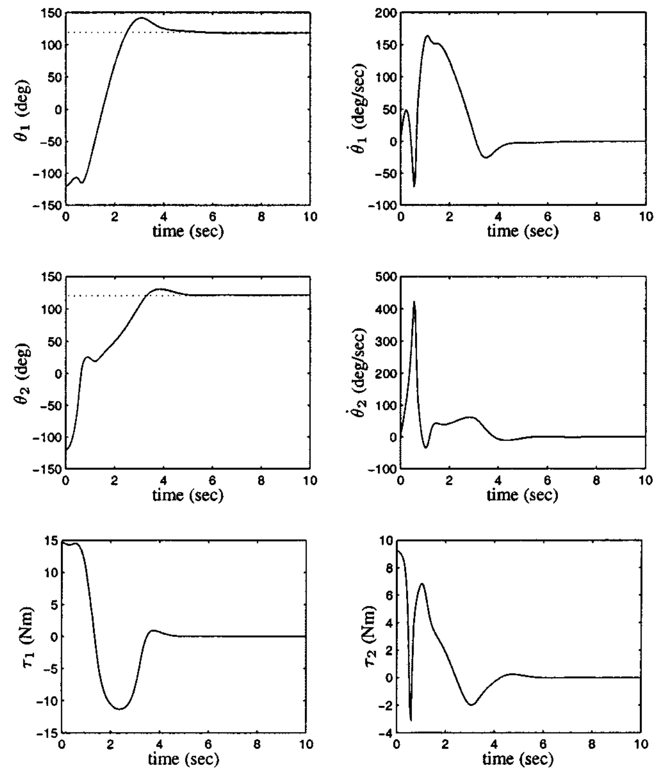

Fig. 15. Result for the initial state inside the training set. The state is initially $\theta_1[0] = -120$ °, $\dot\theta_1[0] = 0$ °/s, $\theta_2[0] = -120$ °, and $\theta_2[0] = 0$ °. The dotted lines represent the reference inputs θ_{1d} and θ_{2d} to the controller.

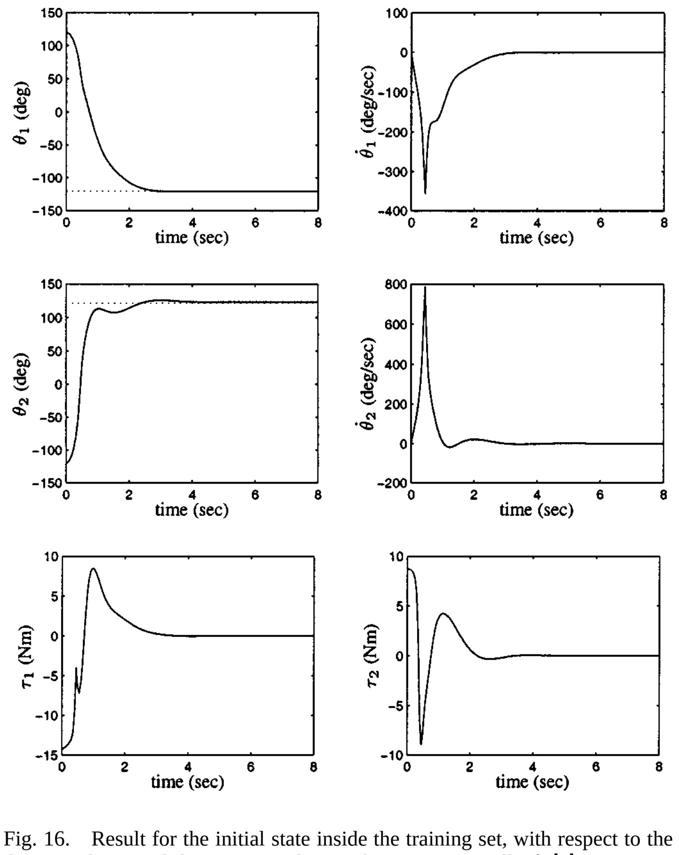

Fig. 16. Result for the initial state inside the training set, with respect to the directionalities and dynamic couplings. The state is initially $\theta_1[0] = 120$ °, $\dot\theta_1[0] = 0$ °/s, $\theta_2[0] = -120$ °, and $\theta_2[0] = 0$ °/s. The dotted lines represent the reference inputs θ_{1d} and θ_{2d} to the controller.

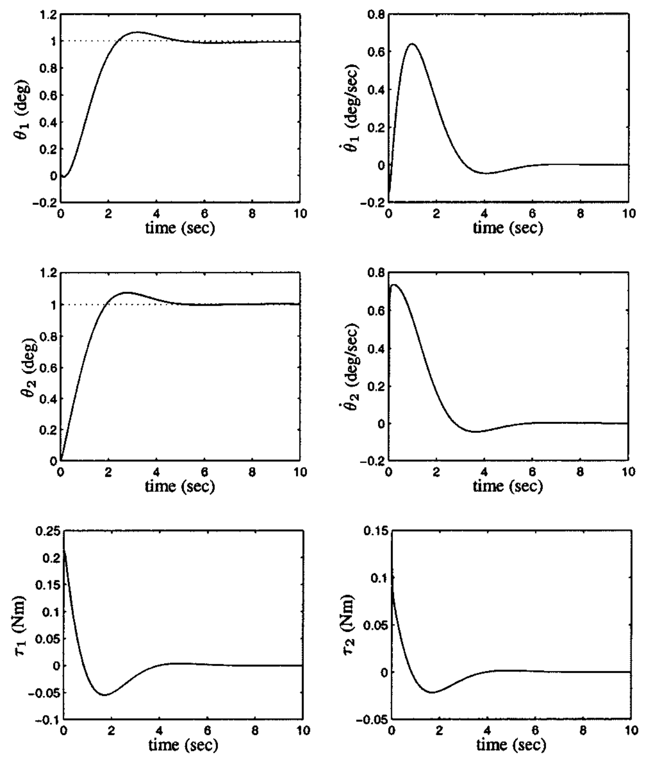

Fig. 17. Result for the initial state inside the training set. The state is initially $\theta_1[0] = 0$ °, $\dot\theta_1[0] = 0$ °/s, $\theta_2[0] = 0$ °, and $\dot\theta_2[0] = 0$ °/s. The dotted lines represent the reference inputs θ_{1d} and θ_{2d} to the controller.

Figs. 18 and 19 show how the neural controller works for the initial states outside the training set. Fig. 18 shows a big swing of the robot-arm around the state space. We initially set the state to $\theta_1[0] = 150$ °, $\dot\theta_1[0] = 0$ °/s, $\theta_2[0] = 150$ °, and $\dot\theta_2[0] = 0$ °/s. Then the controller successfully drives the states to $\theta_1 = -150$ °, $\dot\theta_1[0] = 0$ °/s, $\theta_2 = -150$ °, and $\theta_2 = 0$ °/s, obeying the system's command. Fig. 19 illustrates how the controller handles nonzero initial velocities of the robot-arm. We set the initial state to $\theta_1[0] = -45$ °, $\dot\theta_1[0] = -60$ °/s, $\theta_2[0] = -45$ °, $\dot\theta_2[0] = -60$ °/s. In other words, the robot-arm initially swings around with 0 rpm. Then the controller brings the state to the origin, as we command.

Fig. 20 shows robustness test results of the neural controller. We apply the controller to another two-link robot-arm possessing the following system parameters:

$$\begin{aligned} m_1 &= 1.2\ \text{kg} \\ m_2 &= 2.4\ \text{kg} \\ m_3 &= 0\ \text{kg} \\ l_1 &= 1.2\ \text{m} \\ l_2 &= 1.44\ \text{m} \\ r_1 &= 0.6\ \text{m} \\ r_2 &= \frac{l_2}{2} + \frac{0.5 m_3 l_2}{m_2 + m_3}\ \text{m} \\ I_{z1} &= 0.114\ \text{kg} \cdot \text{m}^2 \\ I_{z2} &= \frac{m_2 l_2^2}{12} + m_2\left(r_2 - \frac{l_2}{2}\right)^2 + m_3 (r_2 - l_2)^2\ \text{kg} \cdot \text{m}^2. \end{aligned}$$

All the parameter values of the new (or perturbed) robot-arm are at least 20% larger than the corresponding parameter values

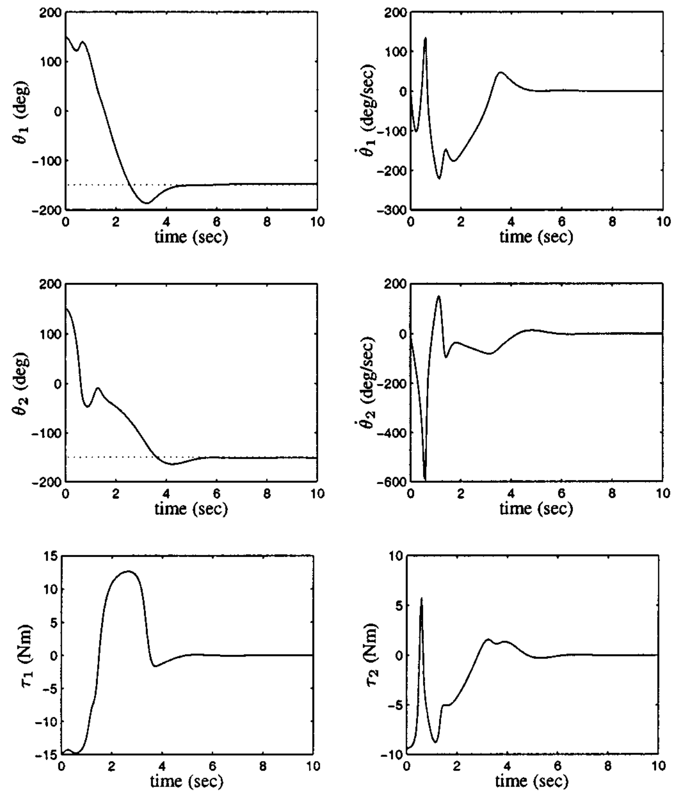
Fig. 18. Result for the initial state outside the training set. The state is initially $\theta_1[0] = 150$ °, $\dot{\theta}_1[0] = 0$ °/s, $\theta_2[0] = 150$ °, and $\dot{\theta}_2[0] = 0$ °/s. The dotted lines represent the reference inputs θ_{1d} and θ_{2d} to the controller.

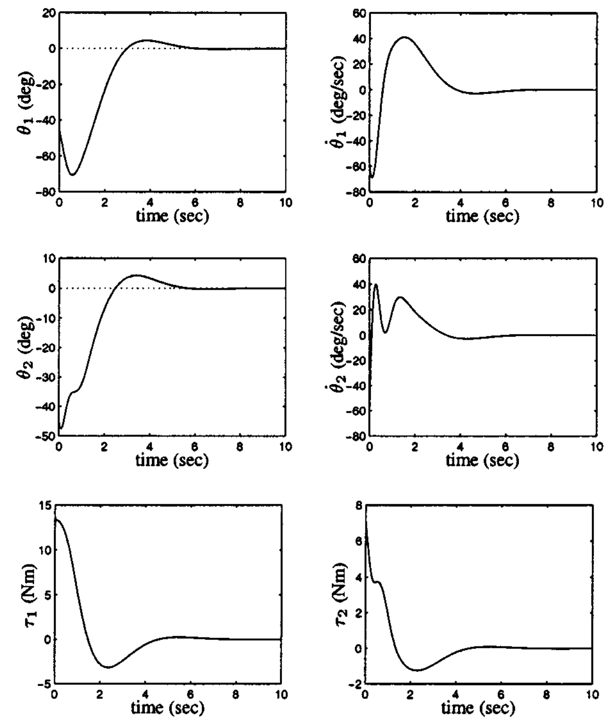
Fig. 19. Result for the initial state outside the training set. The state is initially $\theta_1[0] = -45$ °, $\dot{\theta}_1[0] = -60$ °/s, $\theta_2[0] = -45$ °, and $\dot{\theta}_2[0] = -60$ °/s. The dotted lines represent the reference inputs θ_{1d} and θ_{2d} to the controller.

of the original one that the neural controller is trained for (note that inertial tensors vary with the square of the lengths of links).

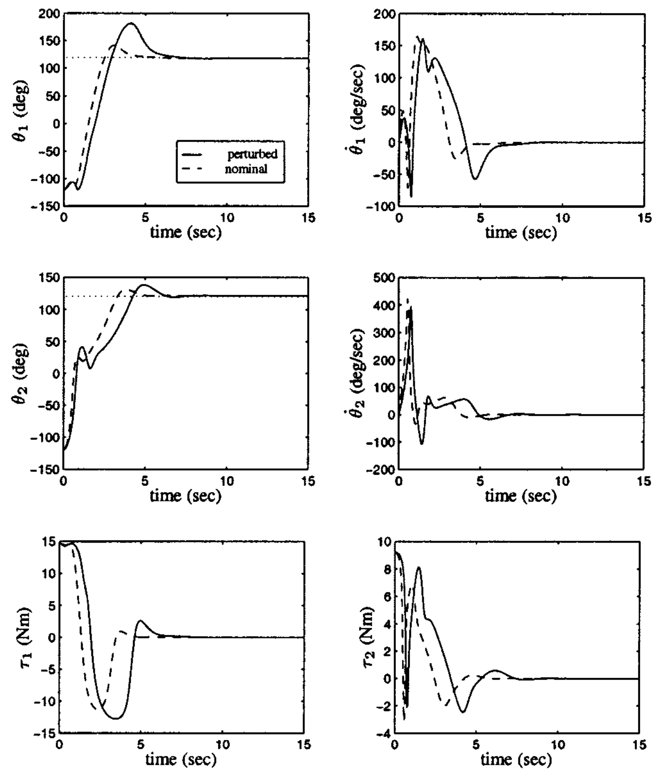

Fig. 20. Robustness tests of the neural controller against model uncertainties. The states are initially $\theta_1[0] = -120$ °, $\dot{\theta}_1[0] = 0$ °/s, $\theta_2[0] = -120$ °, and $\dot{\theta}_2[0] = 0$ °/s. The dotted lines represent the reference inputs, θ_{1d} and θ_{2d}, to the controller for both robot-arms.

In other words, the neural controller has at least 20% parametric model uncertainties with respect to the perturbed robot-arm. We initially set the state of the perturbed system to $\theta_1[0] = -120$ °, $\dot{\theta}_1[0] = 0$ °/s, $\theta_2[0] = -120$ °, and $\dot{\theta}_2[0] = 0$ °/s. Then we command the controller to drive the state to $\theta_1[0] = 120$ °, $\dot{\theta}_1[0] = 0$ °/s, $\theta_2[0] = 120$ °, and $\dot{\theta}_2[0] = 0$ °/s. Fig. 20 shows the state trajectories of the perturbed system, in comparison with those of the the original (or nominal) system; the solid and the dash-dotted lines represent the trajectories of the perturbed and the nominal robot-arms, respectively. Although the perturbed robot-arm responds slower and employs more control efforts than the nominal one because the former is larger and heavier, the controller successfully drives the perturbed robot-arm from given initial states to desired steady states, as it is commanded.

This example demonstrates that NDO works well for nonlinear MIMO control problems. NDO properly handles the directionalities as well as the nonlinearities of nonlinear MIMO systems such as a two-link robot. The example also illustrates that neural controllers NDO produces can be *robust* to the model uncertainties arising in applications.

VI. Conclusion

We present NDO as a practical method for solving dynamic programming problems. NDO enables neural networks to approximate the optimal feedback solutions whose existences DP justifies. Combining the positive features of both methodologies, NDO inherits its practicality from neural networks and its generality from optimal control theory. NDO, however, has two potential drawbacks. First, the NDO solution is not a complete DP solution: it approximates the optimal solution. Local as well as global optima are possible. Its domain of attraction can be limited. Second, the stability of the weight update cannot be guaranteed because its analytical condition has not been developed. In practice, however, these two drawbacks can be overcome by retraining the neural network with different values of its update (i.e., learning) rate or initial weights.

NDO has been demonstrated with several applications including control of autonomous vehicles and of a robot-arm. These applications show that NDO finds — with a reasonable amount of computation and storage — optimal feedback solutions to nonlinear MIMO control problems that would be very difficult to implement in real time with DP. In addition, NDO is applicable by using a neural emulator (an approximate system model) if the mathematical model of a controlled system is not available. NDO can overcome model uncertainties resulting from the discrepancy between a true system and the neural emulator. Moreover, the applications show that the neural controllers NDO produces are able to minimize the effects of disturbances and uncertainties that may arise in real-world applications.

REFERENCES

[1] C. Y. Seong and B. Widrow, "Neural dynamic optimization for control systems—Part I: Background," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 482–489, Aug. 2001.

[2] ——, "Neural dynamic optimization for control systems—Part II: Theory," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 490–501, Aug. 2001.

[3] A. E. Bryson, *Control of Spacecraft and Aircraft*. Princeton, NJ: Princeton Univ. Press, 1994.

[4] ——, *Dynamic Optimization*. Menlo Park, CA: Addison-Wesley-Longman, 1999.

[5] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. Neural Networks*, vol. 2, New York, June 1990, pp. 21–6.

[6] ——, "The truck backer-upper: An example of self-learning in neural networks," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, Washington, D.C, June 1989, pp. 357–363.

[7] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.

[8] D. Rumelhart and J. McClell, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, vol. 1.

[9] J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[10] G. L. Plett, "Adaptive inverse control of plants with disturbances," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1998.

Chang-Yun Seong received the B.S. and M.S. degrees in aerospace engineering from Seoul National University, Seoul, Korea, in 1990 and 1992, respectively, and the M.S. degree in electrical engineering and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, in 1998 and 2000, respectively.

He was a Research Engineer with the Systems Engineering Research Institute (SERI/KIST), Taejon, Korea, before he pursued his graduate studies at Stanford University. He is currently a Research Engineer with Fineground Networks, Campbell, CA. His research interests include neural networks, optimization, control systems, and digital signal processing.

Dr. Seong is a member of AIAA. He was a recipient of the Outstanding Teaching Assistant Award from the AIAA Stanford Chapter in 1997.

Bernard Widrow (M'58–SM'75–F'76–LF'95) received the S.B., S.M., and Sc.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, in 1951, 1953, and 1956, respectively.

He joined the MIT faculty and taught there from 1956 to 1959. In 1959, he joined the faculty of Stanford University, Stanford, CA, where he is currently Professor of Electrical Engineering. He is Associate Editor of several journals, and is the author of about 100 technical papers and 15 patents. He is coauthor of *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall, 1985) and *Adaptive Inverse Control* (Englewood Cliffs, NJ: Prentice-Hall, 1996). A new book, *Quantization Noise*, is in preparation.

Dr. Widrow is a Fellow of AAAS. He received the IEEE Centennial Medal in 1984, the IEEE Alexander Graham Bell Medal in 1986, the IEEE Neural Networks Pioneer Medal in 1991, the IEEE Signal Processing Society Medal in 1998, the IEEE Millennium Medal in 2000, and the Benjamin Franklin Medal of the Franklin Institute in 2001. He was inducted into the National Academy of Engineering in 1995, and into the Silicon Valley Engineering Council Hall of Fame in 1999. He is a Past President and currently a Member of the Governing Board of the International Neural Network Society.