01 Jan 2000

Neurocontroller Alternatives for "Fuzzy" Ball-and-Beam Systems with Nonuniform Nonlinear Friction

Danil V. Prokhorov

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Paul H. Eaton

Neurocontroller Alternatives for "Fuzzy" Ball-and-Beam Systems with Nonuniform Nonlinear Friction

Paul H. Eaton, Danil V. Prokhorov, *Member, IEEE*, and Donald C. Wunsch II, *Senior Member, IEEE*

Abstract—The ball-and-beam problem is a benchmark for testing control algorithms. In the World Congress on Neural Networks, 1994, Prof. L. Zadeh proposed a twist to the problem, which, he suggested, would require a fuzzy logic controller. This experiment uses a beam, partially covered with a sticky substance, increasing the difficulty of predicting the ball's motion. We complicated this problem even more by not using any information concerning the ball's velocity. Although it is common to use the first differences of the ball's consecutive positions as a measure of velocity and explicit input to the controller, we preferred to exploit recurrent neural networks, inputting only consecutive positions instead. We have used truncated backpropagation through time with the node-decoupled extended Kalman filter (NDEKF) algorithm to update the weights in the networks. Our best neurocontroller uses a form of approximate dynamic programming called an adaptive critic design. A hierarchy of such designs exists. Our system uses dual heuristic programming (DHP), an upper-level design. To our best knowledge, our results are the first use of DHP to control a physical system. It is also the first system we know of to respond to Zadeh's challenge. We do not claim this neural network control algorithm is the best approach to this problem, nor do we claim it is better than a fuzzy controller. It is instead a contribution to the scientific dialogue about the boundary between the two overlapping disciplines

Index Terms—ACD, adaptive critic, control, DHP, dynamic programming, extended Kalman filter, fuzzy, HDP, neural networks, neurocontrol, recurrent neural network, time-delay neural network.

I. INTRODUCTION

AT THE World Congress on Neural Networks, 1994, and later, at the IFAC 13th Triennial World Congress in San Francisco, 1996, Prof. L. Zadeh issued a challenge to the neural-network community. This challenge was as follows [1]:

> A more complex problem involving dynamic motion planning—a problem which does not lend itself to solution by the methods of classical control—is what might be called the fuzzy ball and beam problem. The main

difference between this problem and the standard ball and beam problem is that the beam is assumed to be covered with a strip of fuzzy material, e.g., a strip of thick-pile rug. The fuzziness of the beam's surface is intended to preclude the possibility of setting up the differential equations governing the ball's motion (gliding and rolling) on the beam. This rules out—or at least makes it very difficult—to use classical control techniques to derive a way of stabilizing the ball within a prescribed interval. The same obstacle stands in the way of application of neural network and genetic algorithm techniques since a prerequisite to the use of these techniques is the possibility of computer simulation.

(He provided another example fuzzy material, sticky tape, in his earlier plenary talk at World Congress on Neural Networks, 1994.) Zadeh asserted that the only way to control this system was with fuzzy logic, or at least that other approaches would be difficult. We demonstrate two neurocontrollers to solve this problem. Our approach is to train neurocontrollers on a model of the system off-line, and then deploy them with fixed weights for testing on the actual system on-line. First we verify that this approach is valid using the ordinary beam, i.e., one providing uniform friction for the ball everywhere. Then we show that our approach is applicable to the "fuzzy" beam as well.

Our experience with this problem shows that the presence of a sticky surface does increase the difficulty of control, as asserted in the challenge. A shallow beam angle can allow the ball to become stuck on an undesired location, while a steep angle will cause it to shoot past the desired point. Setting up differential equations to describe the ball's dynamics is very difficult since these responses are nonlinear and ill-posed. (On a nonsticky beam, these problems do not exist because the ball's movement is smooth and predictable.) Yet, we believe that modeling this problem is not impossible. In fact, the sheer success of our experiments confirms feasibility of relatively simple control solutions attained with *qualitatively* accurate models.

We begin by describing the experimental apparatus, neural-network architecture, and training algorithms (Sections II and III). In Section IV, we discuss the creation of a model of the system for off-line simulation and training. In Section V-A, a conventional neurocontroller is trained, using backpropagation through time, to balance the ball. Adaptive critic approaches to train neurocontrollers are discussed in Section V-B. We also provide references in that section to papers with full implementation details of these algorithms. In Section VI, we present results from our test on ordinary (nonsticky) and "fuzzy" (sticky)
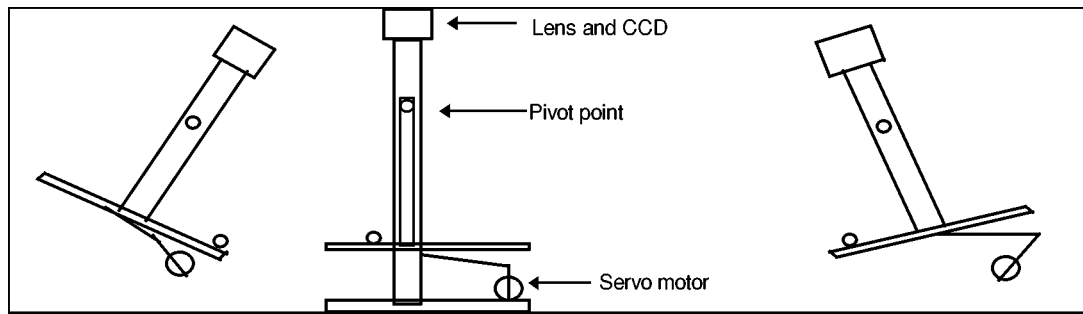
Fig. 1. Hardware setup. The beam moves like a cradle, with a pivot point high above the beam.

beams for both controllers, shown in computer simulations and actual hardware tests.

II. INSTALLATION FOR EXPERIMENTS

In the usual designs for the ball-and-beam, the position sensors are mounted on the beam [7], [20], [21]. For instance, photodiodes mounted along the beam are used as position sensors in [7]. When the ball rolls, it blocks an external light incident on a photodiode thereby signaling its relative position on the beam. We cover the beam with a nontransparent sticky tape, but this obviously would interfere with photosensors as described above. Therefore we built the ball-and-beam setup with displaced fulcrum as shown in Fig. 1. The beam moves like a cradle, with a pivot point high above it. A ball position sensor is moved above the beam such that the ball is always easily detectable, whether the beam is covered with a sticky tape or not.

The displaced fulcrum design just described does make the physics of the problem easier than that of the usual ball-and-beam. The version with nonlinear friction (beam partially covered with sticky tape) thus inherits this advantage. However, our design retains the intrinsic instability of balancing the ball *in an arbitrary off-center location* on the beam as the essential property of the usual ball-and-beam problems. In any case, it is a necessary step, enabling experimentation with the "fuzzy" ball-and-beam, since the nonlinear friction is obtained by covering the surface where photodetectors would be used in other implementations.

The ball position sensor is a 512-element linear charge-coupled device (CCD). The light reflecting off the ball and beam is focused onto the CCD using a lens. The ball is painted black and the beam is white. The software reads in the CCD pixels serially, with sampling period of 0.055 s, looking for the dark spot and determining the ball's position on the beam.

A servo motor, using pulsed width modulation (PWM) encoding, moves the beam. The PWM signal is a digital signal with a fixed frequency (the same as the inverse of the sampling period) and a variable duty cycle, with changes in the duty cycle changing the servo motor position.

The ball is 2 cm in diameter. The controller can balance it at any position on the beam, which is 40 cm long. A foam bumper is at each edge of the beam. The CCD reads in the beam from pixel 189 to 412, allowing for 223 pixels to scan the 40-cm beam. This allows an 0.18-cm accuracy limit in determining the ball's position. Actual accuracy is 0.54 cm after experiments
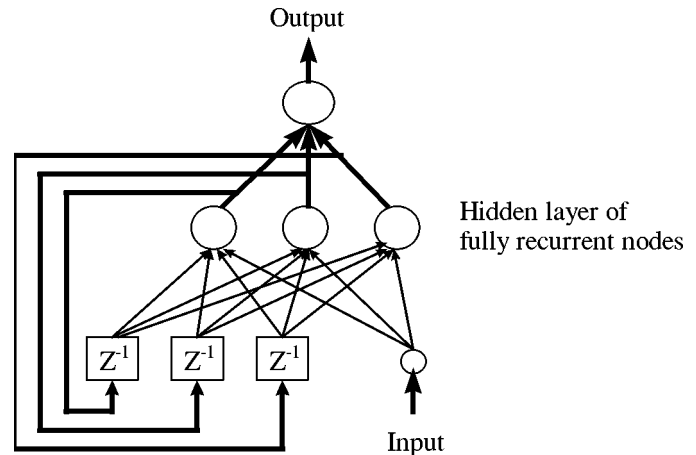

Fig. 2. Recurrent network architecture. The output of each neuron in the hidden layer from the previous time step is used as input for the current step.

showed an added ±1 pixel from system noise. The servo motor can be set to any value from zero to 255. Due to hardware limitations, only the range between 50 and 140 is used to move the beam. This range produces 90 different possible positions, which set the beam at an angle between 79–103°, with 90° corresponding to the beam being leveled. This allows for the beam to be set to positions in 0.26° increments.

The "fuzzy" beam used in this paper was created by laying a sticky tape across half of the beam (sticky surface facing upward). One half of the beam (approximately −1 to zero in normalized units, and 189 to 290 in the hardware) remained the same as the ordinary beam, while another half (zero to +1 in normalized units, and 290 to 405 in the hardware) was covered with the tape.

III. NEURAL NETWORKS USED AND THEIR TRAINING ALGORITHM

Our training method is backpropagation through time truncated with depth h [BPTT(h)] with node-decoupled extended Kalman filtering (NDEKF). BPTT(h) uses h copies of the network with the same weights but different node activations corresponding to different time steps. We typically used $h = 10$. The recurrent neural networks (RNN's) we use (Fig. 2) are discrete-time recurrent multilayer perceptrons [5].

In BPTT(h), multiple copies of the same network weights are made. This can be pictured as one larger network, with each time step representing a hidden layer passing its output to the next

layer. While the copies' weights are usually the same, the node activations are different and time dependent. Thus, unfolding the network provides a natural way to preserve this time dependence. Starting at the beginning of the data, each copy calculates its outputs. At the end, the error is calculated for the last copy, and the weights are updated using the chain rule for ordered derivatives. The copies are then moved forward one time step and the process is repeated [14]–[16].[1]

EKF training is a parameter identification technique for an RNN [4], which adapts weights of the network pattern-by-pattern, accumulating training information in approximate error covariance matrices and providing individually adjusted updates for the network's weights.

All weights of the RNN are assembled in a vector W of length M. This is split into G disjoint groups. W_i is a vector of the ith group of weights. $W_1 \cup W_2 \cup \cdots \cup W_G = W$, and size($W_1$)+ size($W_2$) + $\cdots$ + size(W_G) = M.

The EKF method requires that we compute derivatives of the RNN's N outputs, rather than output errors, with respect to the weights. These derivatives are obtained using BPTT(h) [16]. We store them in a set of matrices H_i, where each H_i has dimensions size(W_i) $\times N$.

The following equations form the basis of the EKF training procedure:

$$A(k) = \left(\frac{I}{\eta(k)} + \sum_{i=1}^{G} H_i^T(k) P_i(k) H_i(k) \right)^{-1} \qquad (1)$$

$$K_i(k) = P_i(k) H_i(k) A(k) \qquad (2)$$

$$W_i(k+1) = W_i(k) + K_i(k) e(k) \qquad (3)$$

$$P_i(k+1) = P_i(k) - K_i(k) H_i^T(k) P_i(k) + Q_i(k) \qquad (4)$$

where

$\eta(k)$	scalar learning rate;
$K_i(k)$	Kalman gain matrix for the ith group of weights;
$e(k) =$ $d(k) - y(k)$	$N \times 1$ error vector;
$d(k)$	vector of the desired outputs;
$y(k)$	actual output vector of the RNN.

$e^T(k) e(k)/2$ forms the squared error, $P_i(k)$ is the size(W_i) $\times$ size(W_i) approximate error covariance matrix which models correlation between each pair within the ith group of weights, and $Q_i(k)$ is a positive diagonal matrix that helps avoid numerical divergence of the procedure and prevents getting stuck in a local minimum [4].

Grouping of weights can be done in a variety of ways. We employ grouping by node, i.e., weights belonging to the same

neuron are grouped together. Thus, we ignore frequently unnecessary correlation between weights belonging to different neurons. This results in a significant reduction of computational complexity and storage requirements since the size of each error covariance matrix P_i can then be made much smaller than M^2, the size in the case when $G = 1$.

The matrices $P_i(0)$ are initialized as diagonal matrices with large diagonal elements (values between 100 and 10 000). User-specified values of $\eta(k)$ are usually increased from 0.01 to one whereas diagonal components of $Q_i(k)$ are decreased from 0.01 to 10^{-6} as training progresses.

The set of matrices H_i in (1), (2), and (4) is obtained by truncated backpropagation through time with depth ten. This means that we do not use more than ten copies of the network to accumulate appropriate derivatives in the matrices H_i, hence the notation BPTT(10). Training usually lasts for around 100 passes, where one pass corresponds to a complete processing of the whole training set.

IV. SYSTEM SIMULATION

Our approach to balancing the ball on the beam belongs to the category of indirect adaptive control [2]. Problems of this category are characterized by the presence of a system model, which simulates behavior of the actual system. The model naturally has to be accurate enough in predicting the system behavior over a sufficiently long time horizon. The length of the time horizon depends on the training mode. If training is performed on-line, the model is typically used for short-term predictions of the system outputs. For example, in the case of one-step-ahead predictions the model gets outputs of the actual system from the previous time step as its inputs. The model outputs are predictions of the current system outputs. Such model-system configuration is known under different names including series-parallel model [3] and teacher forcing [15]. We model the ball-and-beam system via an NN henceforth referred to as an ID (identification) network. After this network is trained, its weights are fixed, and it *replaces* the system in subsequent off-line training of the neurocontrollers. We emphasize that the complete replacement of the actual system is performed. Our configuration is therefore an extreme case of so-called parallel model [3]. Unlike the series-parallel model, it features an ID network inputting its own predictions for the previous time step instead of the system outputs. The parallel model is used when the ID network is to simulate long-term behavior of the system, which is precisely what we need. The parallel method clearly puts more demands on the NN than the series-parallel method. This is justified by the convenience of off-line training, which precludes the latter.

Regardless of whether the parallel or series-parallel model is used, the main function of the model for training controllers is to provide sensitivity signals, i.e., gradients of outputs with respect to inputs. These gradients are used to adapt adjustable parameters of the controller. It is convenient to use the backpropagation algorithm to obtain the gradients,

---

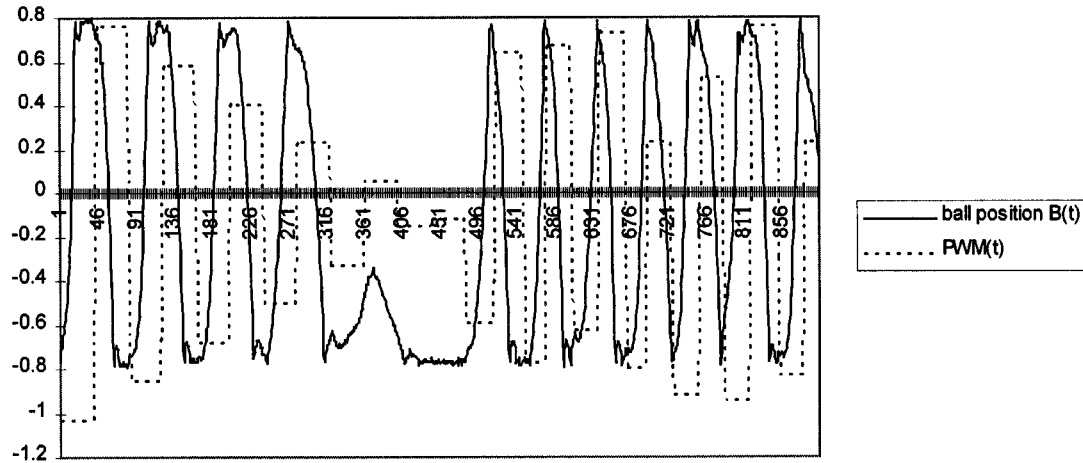[1]Our notation BPTT(h) corresponds to BPTT($h - 1$) of [15] and [24].

Fig. 3. Example training data set. The first 350 steps are a predetermined pattern executed moving the beam through its entire range of motion. The next 650 steps are randomly generated PWM values and the corresponding ball positions.
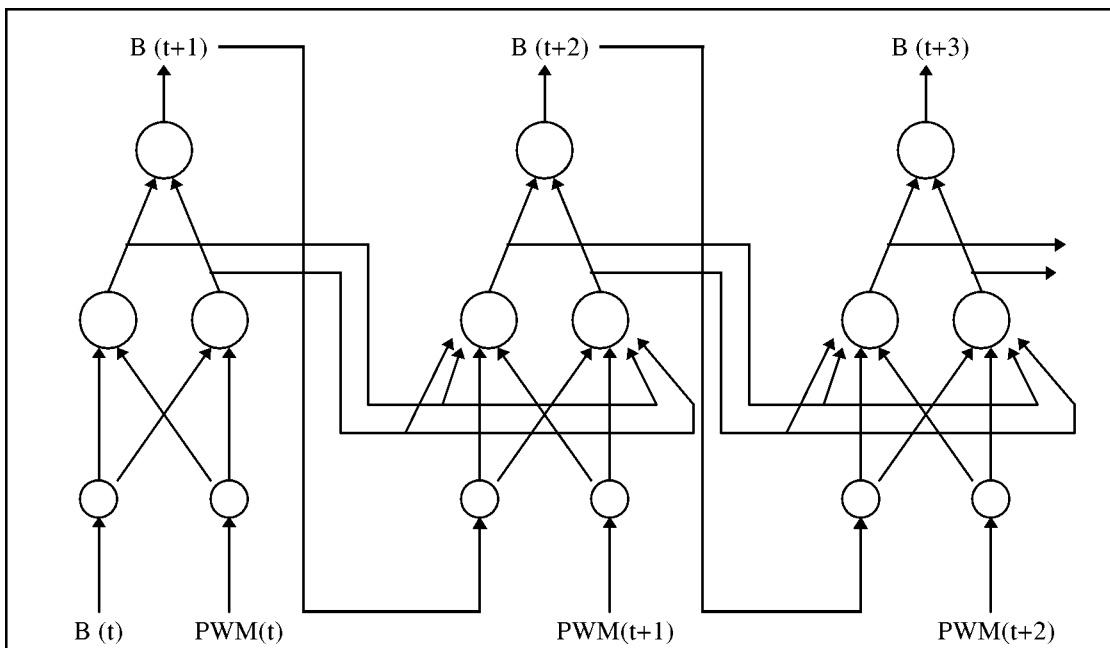

Fig. 4. Architecture for the parallel model. The ID network uses its previous prediction of the ball position and the PWM signal from a data file as its inputs. This creates a reliance on the accuracy of previous predictions in the current position calculation.

in which case one recovers backpropagation through the forward model as proposed in [8].

Our ID network uses two inputs, the ball position, $B(t)$, and the PWM, PWM(t), and computes the ball position $B(t+1)$. It is a one-hidden-layer RNN. The hidden layer has two neurons which read in the inputs and the recurrent values from the previous time, and pass their outputs to one output neuron.

About 40 data files were created for training the ID network using various techniques such as a human randomly changing the angle, a human trying to balance the ball, the computer randomly changing the beam angles, and the computer changing the beam angle in specific patterns. The best models were created using a combination of the last two. In the beginning of gathering the data file, the computer was moving the beam from the largest angle in one direction, followed by the largest angle

in the other direction. The beam then oscillated back and forth, slowly decreasing the angle until the beam was horizontal. This portion of the data file was followed by the computer randomly choosing a beam angle and staying in this position for a certain amount of time. An example of a typical data set recorded in this way is shown in Fig. 3.

The parallel model shown in Fig. 4 feeds the output from each step as the input for the next time step. Only the first few ball positions are read from a data file of consecutive ball positions and corresponding PWM's providing the initial ball position, after which the network uses its previous estimate of the ball position as the current input. The parallel method is trained in 70 to 100 passes using BPTT(10) with NDEKF as described in Section III. A typical comparison of the actual $B(t)$ and the corresponding predictions of the ID network on the training data is shown in
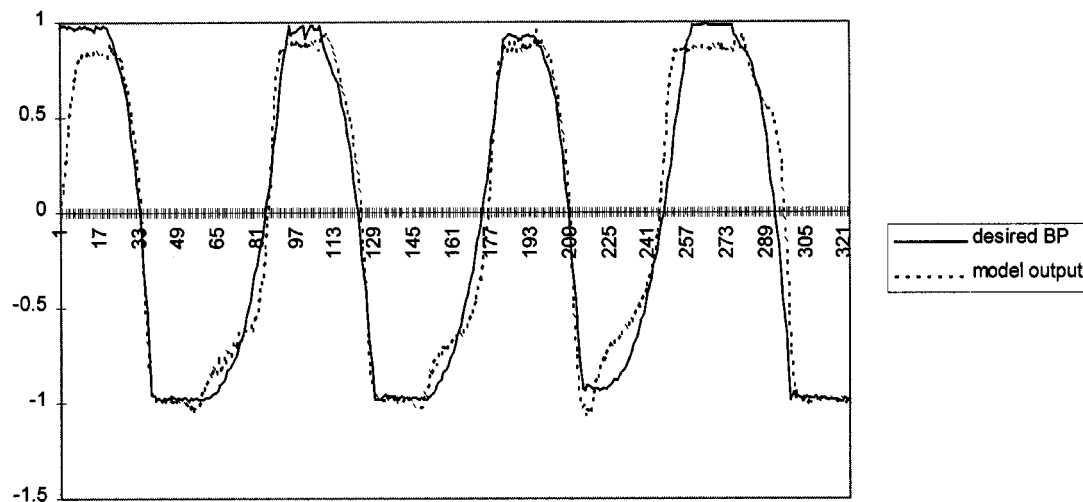
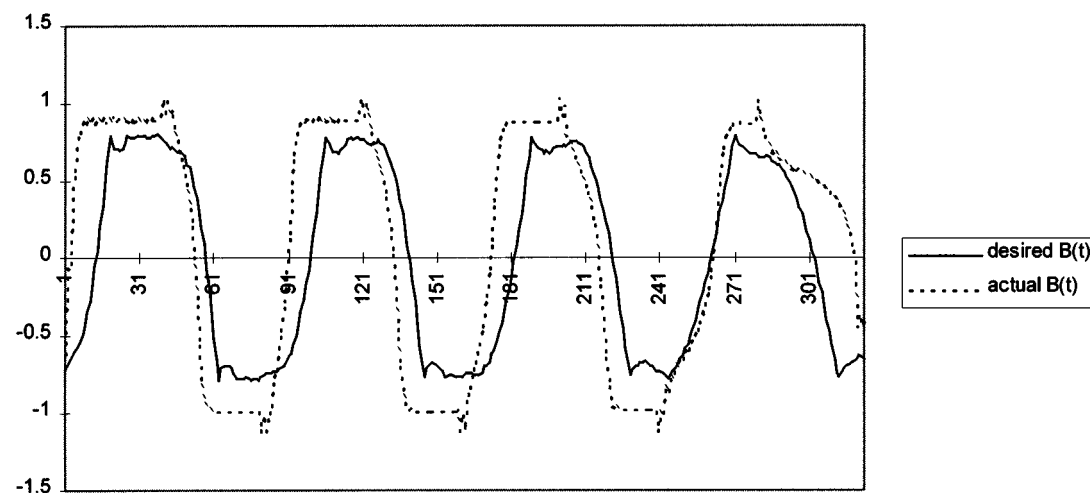Fig. 5. Training results of the ID network trained as the parallel model.

Fig. 6. Test results for the ID network trained as the parallel model. This typical result demonstrates that the ID network indeed captures essential long-term dynamics of the actual ball-and beam system.

Fig. 5. An example test performance of the ID network is illustrated in Fig. 6. While discrepancies between the predicted and the actual $B(t)$ appear to be large, they turn out to be small enough to enable acceptable off-line training of our neurocontrollers, to be discussed in Sections V and VI. More details on the system simulation using series-parallel and parallel models are given in [17] and [18].

V. NEUROCONTROLLERS

Once we have the ID network, we can proceed with designing and training of the system's controller. Section V-A discusses the conventional neurocontroller, i.e., one which is trained using BPTT with NDEKF in the closed loop with the ID network. In Section V-B, we discuss an adaptive critic-based neurocontroller. To distinguish between the two neurocontrollers we used the designator "conventional" for the former. We will later show that our best performance for this problem was obtained with the adaptive critic known as DHP. We provide some motivation for this design and references to complete implementation details.

A. Conventional Neurocontroller

The controller is a one-hidden-layer RNN, with two nodes in the hidden layer and one node in the output layer. The neurocontroller uses the two inputs, the current ball position (B), and the desired ball position ($Bdesired$), to produce an output which is the PWM signal used by the ID network.

The desired ball position is a value provided in a data file. The current ball position comes from the output of the ID network from the previous time step. The neurocontroller is shown with its connections to the ID network in Fig. 7. The output of the neurocontroller (PWM) is a sigmoid node, where PWM = -1 sets the beam to the highest angle possible in one direction, and PWM = $+1$ moves to the highest angle in the other direction. The controller is trained using BPTT(10) with an error term corresponding to the difference between the current ball position and the desired ball position at each time step. This error is backpropagated through the ID network to the neurocontroller, providing $\partial E(t)/\partial \text{PWM}(t)$. The weights of the controller are
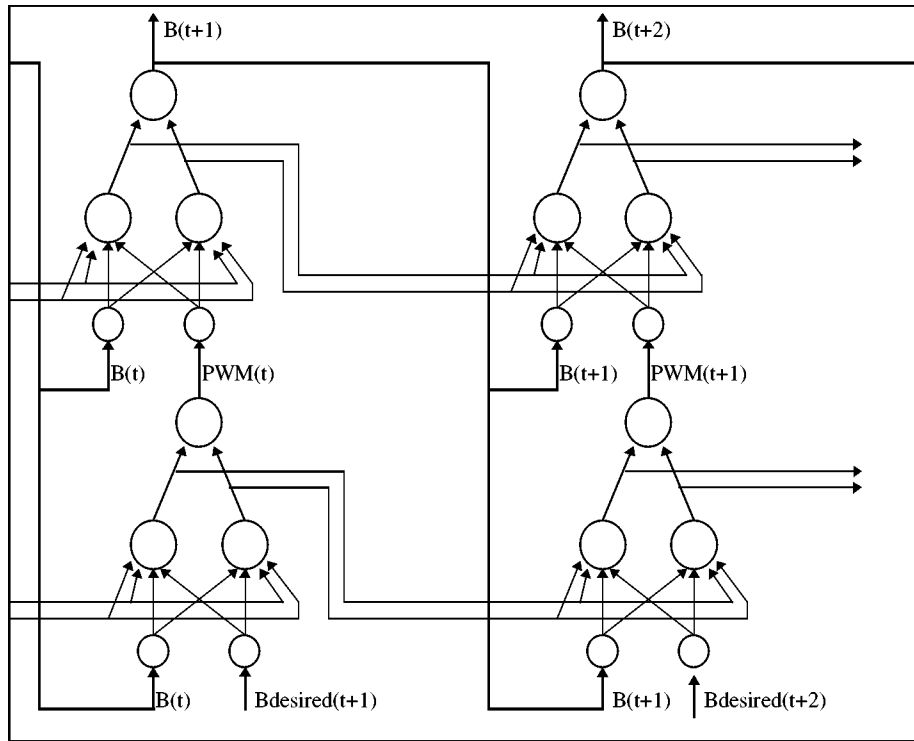
Fig. 7.   Controller network connected to the ID network. Both networks contain two hidden neurons and one output neuron. The controller uses the current $B(t)$ from the previous time step and the desired position $Bdesired(t)$ from a data file, calculating the needed PWM(t) value. The ID network receives the current $B(t)$ and the controllers output PWM(t) value to determine the next ball position $B(t+1)$.

then updated using NDEKF and the appropriate derivatives with respect to the controller weights.

B. Adaptive Critic-Based Neurocontroller

A family of approaches united under the common name adaptive critic designs (ACD's) have recently emerged as a synthesis of reinforcement learning, dynamic programming, and back-propagation [10], [19], [22], [23]. This section discusses the most salient issues of ACD's with respect to this problem. A typical ACD consists of the critic network, the ID network, and the controller network, also called the action network in ACD literature. The critic is trained to estimate the cost-to-go function $J(t)$ of the Bellman equation of classical dynamic programming [13]. Similarly to the off-line training algorithm Dyna [9], in our case the critic is added to the ID network and the conventional controller of Section V-A to obtain an adaptive critic-based neurocontroller. The addition of the critic turned out to be crucial in improving the balancing performance, as shown in Section VI.

The critic is a one-hidden-layer RNN with three hidden neurons and one output neuron. The critic inputs are the actual ball position $B(t)$ and the corresponding desired ball position $Bdesired(t)$. Two ACD's, heuristic dynamic programming (HDP) and dual heuristic programming (DHP), were implemented. HDP outputs the cost-to-go function $J(t)$, while DHP outputs the derivative of $J(t)$. The reader is referred to [11], [12], and [19] for the detailed descriptions of these ACD's. As we shall see, DHP works well for this problem, whereas HDP does not.

We nevertheless begin by introducing the architecture of HDP, because DHP is best explained by its contrast with HDP, and the shortcomings of HDP for this problem serve to motivate the use of DHP. The HDP critic produces the $J(t)$ function described above as the output, calculated using

$$\text{Error}(t) = J(t) - \gamma J(t+1) - U(t) \tag{5}$$

where

$$U(t) = \tfrac{1}{2}(B(t) - Bdesired(t))^2. \tag{6}$$

The error is calculated by finding the output error, $U(t)$, and the $J(t)$ and $J(t+1)$ terms for the current and next time step. By using this cost-to-go function, the effect of the current neuro-controller output on future positions can be seen. For instance, if the beam is set to a very high angle, the ball could roll to the desired position faster than from a small angle, but it will be more difficult to stop the ball when it arrives. Learning to incorporate this tradeoff is why reinforcement learning allows good control decisions in spite of the inability to create an accurate model. This advantageous property of ACD's was also noted in [22].

The term γ in (5) is called the discount factor and is set to a value between zero and one. When γ is close to zero, the critic network will be trained with a very small predictability horizon [if $\gamma = 0$, this horizon is just the current time step error $U(t)$], while having values of γ close to one is equivalent to extending prediction horizon of the critic up to infinity. Once the critic network is trained, its weights are fixed and the critic is used to
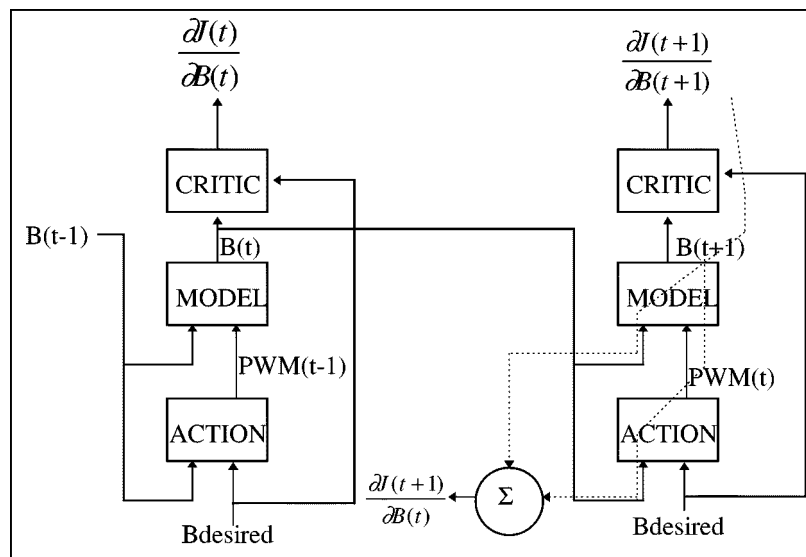
Fig. 8. Diagram of the connections between networks for DHP. This method uses a critic trained to produce a derivative of the cost-to-go function. The derivative of the cost-to-go function is backpropagated through the networks as shown by the dashed lines. [The left branch captures the dependency upon $B(t)$, and is thus the source of error signal in the previous time step. The right branch captures the dependency upon the control signal PWM(t). This enables training of the Action net as shown.] The model (ID) network and the action (controller) network are the same as those in Fig. 7.

train the action network. This is done by using the derivative of the cost-to-go function as the error term

$$0 - \frac{\partial J(t)}{\partial B(t)}. \qquad (7)$$

Simultaneously requiring smooth estimates and a minimal final cost is proven to minimize cost over time [6]. Backpropagation through (now fixed) weights of the critic and model networks yields an error signal for the action network. The training of the action network is performed until its error reaches a minimum, and the weights of the action network are fixed and used to retrain the critic. This process is continually repeated until the neurocontroller performance is no longer improved over the previous training pass. The training process follows guidelines of the general training procedure for ACD's proposed in [19], and the NDEKF algorithm is used for updating both critic and action weights.

For this problem, HDP, one of the simplest ACD's, produced unsatisfactory results. This was expected, because, as we shall explain below, DHP computes derivatives that are essential to training the action network that makes control decisions. To illustrate the limitations of HDP with respect to this issue, we trained the HDP critic using $\gamma = 0$ in (5), i.e.,

$$\text{Error}(t) = J(t) - U(t). \qquad (8)$$

The trained critic showed it could produce results with the error (8) close to zero implying that

$$J(t) \approx U(t). \qquad (9)$$

However, the problem occurred when the derivative term $\partial J(t)/\partial B(t)$ was examined. This derivative term was very inaccurate when compared to $\partial U(t)/\partial B(t)$. But an inaccurate $\partial J(t)/\partial B(t)$ causes improper training of the action network due to (7). The choice of $\gamma = 0$ turns the task of approximating the cost-to-go function $J(t)$ into the much simpler task of recreating the one-step cost function $U(t)$. It is therefore hard

to expect a more accurate $\partial J(t)/\partial B(t)$ for $\gamma > 0$. This inaccuracy of $\partial J(t)/\partial B(t)$ in the most benign case demonstrated that even if a network can produce a close estimate of a desired function, it is still possible for the network to produce a poor approximation of the derivatives of this function, unless it is trained to estimate them explicitly.

The explicit approximation of $\partial J(t)/\partial B(t)$ is precisely the goal of the more sophisticated ACD called dual heuristic programming (DHP). DHP uses a critic whose output tries to minimize

$$\text{Error}(t) = \frac{\partial J(t)}{\partial B(t)} - \gamma \frac{\partial J(t+1)}{\partial B(t)} - \frac{\partial U(t)}{\partial B(t)}. \qquad (10)$$

The advantage of this ACD is that the critic produces the $\partial J(t)/\partial B(t)$ term as its output, to be directly used to adapt the action network [see (7)]. The disadvantage lies in the necessity to perform extra computations for the $\partial J(t+1)/\partial B(t)$ term, but this is a minor technical difficulty. The DHP arrangement of all three networks for two consecutive time steps is shown in Fig. 8. DHP is an algorithm of moderate complexity, and all our experience with ACD's to date recommends it as a good tradeoff between complexity and power. Reference [22] contains implementation details pertaining to DHP, while [19] and [23] give an analysis of the whole family of ACD's, including an algorithm which includes HDP and DHP as special cases. While ACD's are sufficiently complex to preclude even an overview section here, these papers, especially [23], provide the complete discussion of relevant issues.

VI. RESULTS

A. Ordinary (Nonsticky) Ball-and-Beam

1) Conventional Neurocontroller: The conventional neurocontroller was trained using the ID network simulating the ball-and-beam system with the ordinary beam, first around a single position and then around multiple positions. The
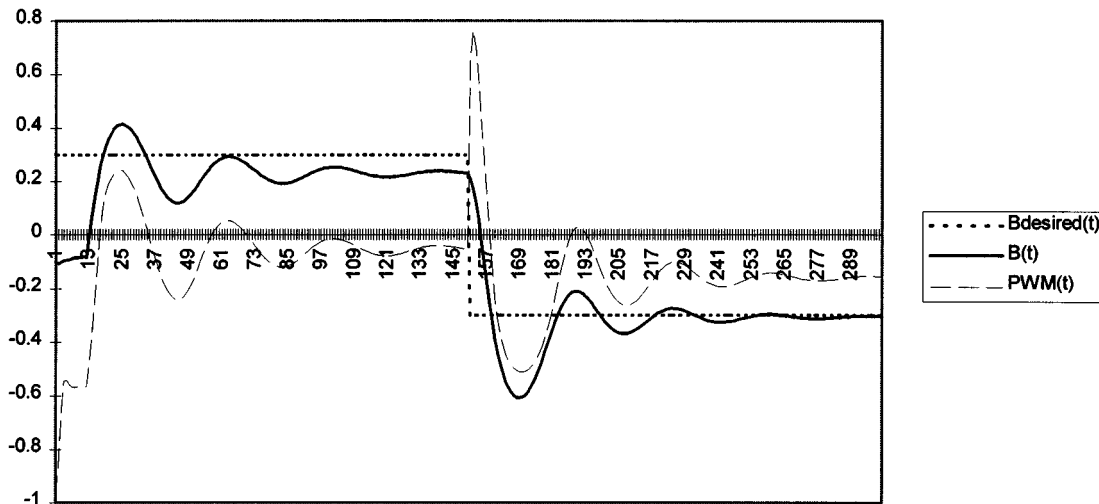
Fig. 9.   Computer simulation of conventional controller test performance on ordinary beam. The results show the conventional neurocontroller approach is capable of moving the ball to different desired positions in a relatively short period of time.
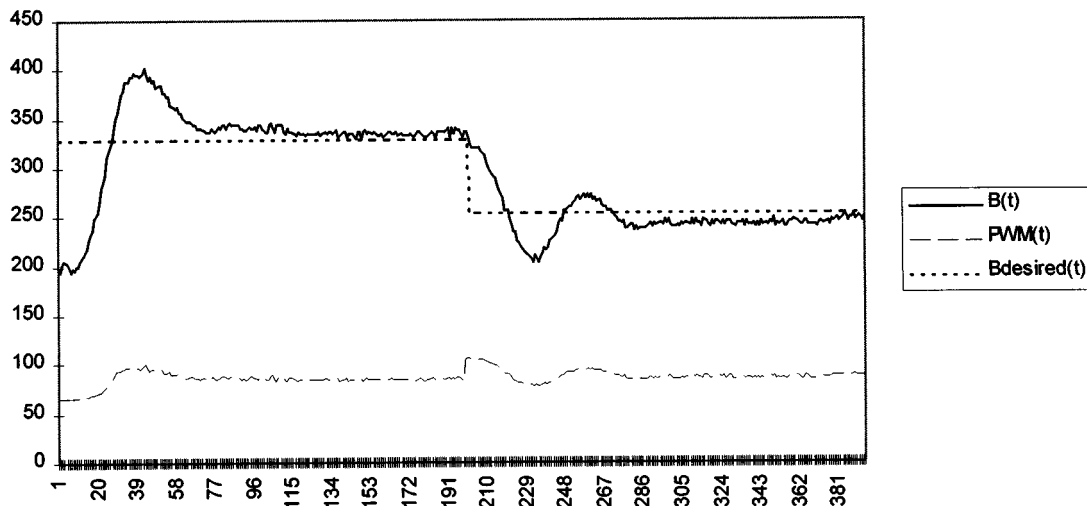

Fig. 10.   Hardware results of conventional controller test performance on ordinary beam. The results show the conventional neurocontroller is able to move and balance the ball around different positions on the beam. These are the same desired balancing points (dotted) as in Fig. 9 but the overall duration is about 100 time steps longer. This difference between the off-line simulations and actual hardware runs is presented in Figs. 11–16.

controller trained relatively fast on the single position, and depending on the distance between the balancing positions, could also train quickly on multiple balancing points. The network was repeatedly trained on many different positions using random initial weights with the best neurocontroller being chosen and applied to test simulations. An example test simulation for this type of controller is shown in Fig. 9. Once it was shown that the neurocontroller could balance the ball in the off-line simulations, the weights were saved and applied to the actual hardware, with the test performance as shown in Fig. 10.

2) Critic-Based Neurocontroller: Neurocontroller training within the DHP framework began with random weights in both the neurocontroller and critic networks. The critic network was trained first, using $\gamma = 0$ in (10) (see Section V-B). This training made the first critic act very similar to the explicit error function used to train the conventional neurocontroller. The critic quickly trained to almost zero error within the first ten passes. Once the error reached a minimum, the critic weights

were saved and the critic was used to train the action network. The results of this were successful, but not as good as the conventional neurocontroller. This was because $\partial J(t)/\partial B(t)$ is an approximation of $\partial U(t)/\partial B(t)$, and there is a slight difference between the two. When the minimum was reached, the first neurocontroller's weights were saved, and the new critic was trained using the first neurocontroller, this time with $\gamma = 0.6$. This process was repeated, with gradually increasing γ (final value 0.9), until the neurocontroller performance was no longer improving. Fig. 11 shows the results of the test simulation for the critic-based neurocontroller and Fig. 12 shows the hardware test results.

3) Comparison of the Two Neurocontrollers Response Characteristics: We observe that the conventional neurocontroller used smaller changes in the PWM signal to move the ball. This makes the controller slower when making a large change in PWM, causing overshooting to be observed in many hardware tests. The conventional neurocontroller appears to have a greater
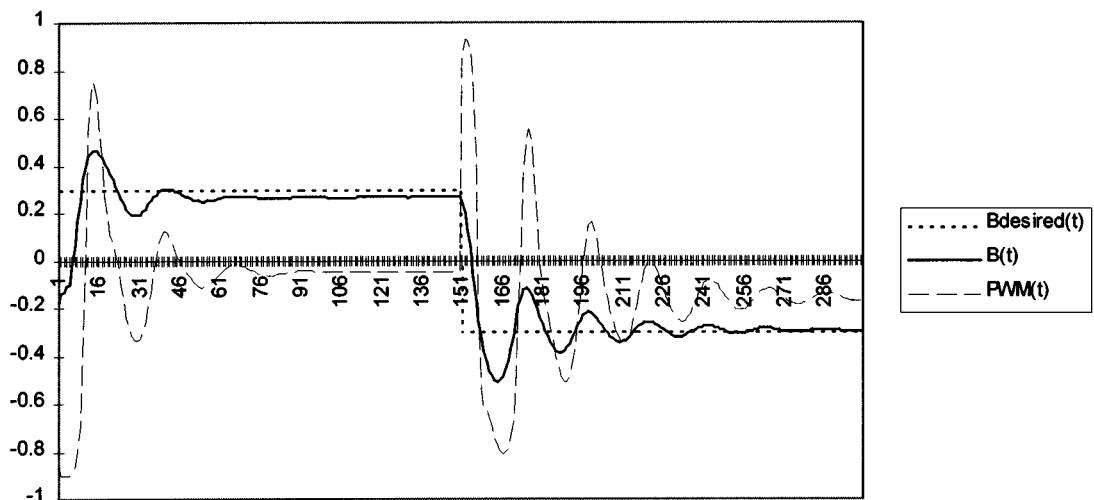
Fig. 11.   Computer simulation of critic-based controller test performance on ordinary beam. The critic based neurocontroller generates larger PWM(t) values compared to the conventional approach when moving the ball. These larger values cause "ringing" effects on the motion of the ball, seen in the second balancing position.
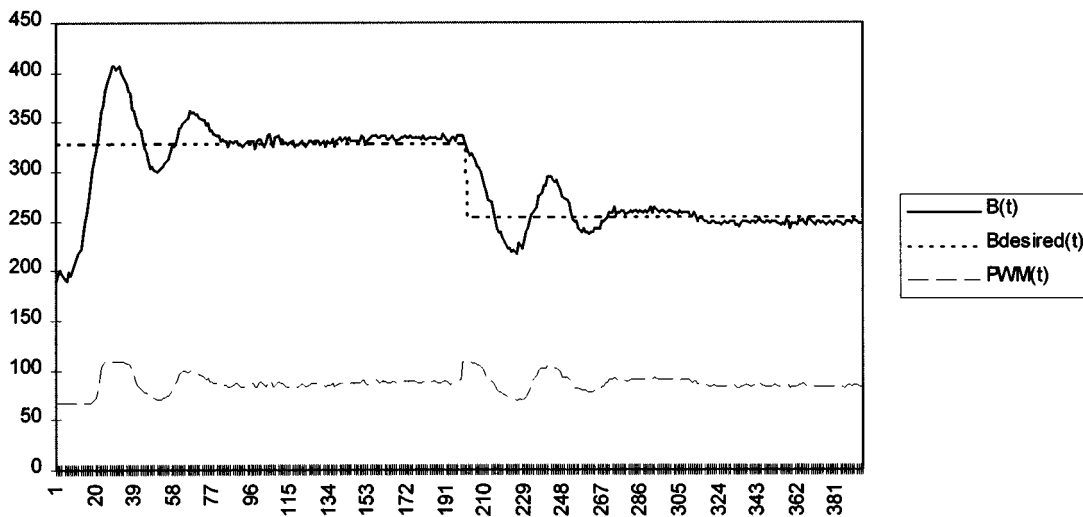

Fig. 12.   Hardware results of critic-based controller test performance on ordinary beam. The hardware results demonstrate the critic based training approach is a viable solution to training neurocontrollers. The larger PWM(t) values generated "ringing" on hardware tests, but reach the final positions in times similar to those of the conventional neurocontroller.

TABLE I
COMPARATIVE RESULTS OF THE NEUROCONTROLLERS FOR THE ORDINARY BALL-AND-BEAM

Neurocontroller	Settling time, sec	Rise time, sec	Overshoot, %
Conventional	5.5	1.4	30
Adaptive critic-based	4.5	1.1	35

difficulty in correcting this than the critic-based neurocontroller. The critic-based neurocontroller used larger PWM signals, usually causing overshoot. However, it could compensate for this quickly resulting in a shorter settling time compared to that of the conventional neurocontroller.

As seen in Figs. 9–12, performances of both controllers are comparable, with small discrepancies between the results of off-line simulations (Figs. 9 and 11) and those of runs on the actual system (Figs. 10 and 12) being observed.

Table I quantifies performance of both neurocontrollers in terms of settling time, rise time, and the overshoot. All numbers represent averages over 20 test trials of balancing the ball.

Thus, we have verified that our approach of training neurocontrollers off-line on the model (ID network) of the actual system is valid when applied to balancing the ball on the ordinary beam. In the next section, we demonstrate that the approach remains applicable to balancing the ball on the "fuzzy" beam as well.
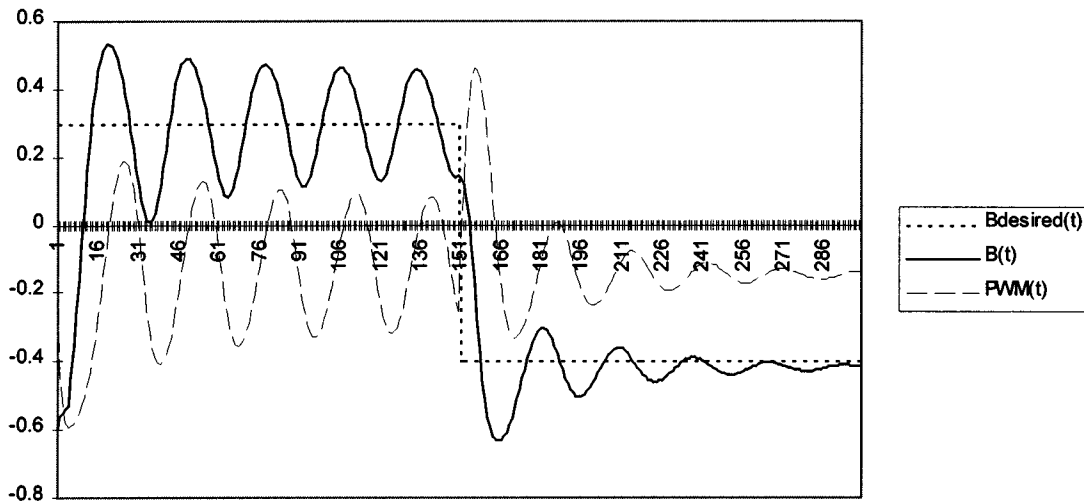
Fig. 13. Computer simulation of conventional controller test performance on fuzzy beam. The conventional neurocontroller training approach was attempted multiple times without producing an adequate controller capable of balancing the ball on the fuzzy beam.
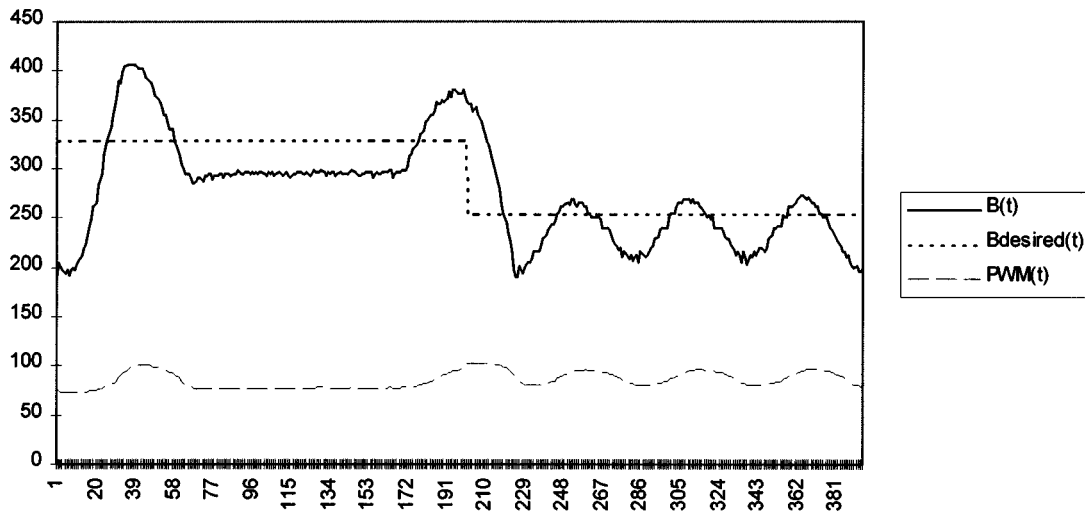

Fig. 14. Hardware results of conventional controller test performance on fuzzy beam. A typical conventional neurocontroller trained on the fuzzy model was applied to the hardware, unable to balance the ball.

B. "Fuzzy" (Sticky) Ball-and-Beam

1) Conventional Neurocontroller: Similar with Section VI-A1, the conventional neurocontroller was trained using the ID network, but this time the ID network was trained in advance to simulate dynamics of the system with the "fuzzy" beam. A typical test simulation of the best neurocontroller is shown in Fig. 13. A typical performance of the same controller when applied to the actual hardware is illustrated in Fig. 14.

2) Critic-Based Neurocontroller: Similar with Section VI-A2, after the critic was trained first (using $\gamma = 0$), critic weights were fixed and used to train the neurocontroller. On the first cycle the controller reached a reasonable minimum of the RMS error term quickly, after two or three passes through a 300-point data file. The critic was then retrained using $\gamma = 0.6$, and used for further training of the controller. This retraining was repeated for several cycles, usually three or four, with the final $\gamma = 0.9$. By this process, the neurocontroller eventually functioned at a performance level that could not be improved by further training. An example test simulation is shown in Fig. 15. This neurocontroller was then applied to the actual system, with the typical results shown in Fig. 16.

3) Comparison of the Two Neurocontrollers' Response Characteristics: Comparing Figs. 13 and 14 and 15 and 16, we conclude that the critic-based neurocontroller outperforms the conventional neurocontroller both in the off-line simulation and hardware tests, but in both cases there are differences between the performance in the simulations and corresponding actual behavior. These differences are more salient than those observed in the case of the ordinary ball-and-beam (see Section VI-A3). In particular, the ball is seen to be stuck in the time interval from 77 to 172 in Fig. 14, whereas it should have oscillated according to Fig. 13. Alternatively, the ball keeps oscillating in the time interval from 229 to the end in Fig. 14 whereas it should have settled down according to Fig. 13.
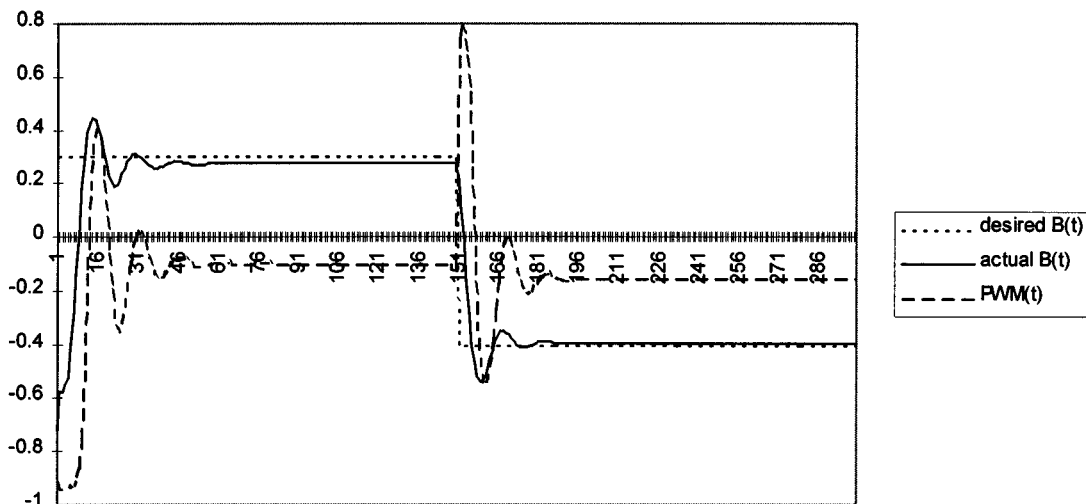
Fig. 15. Computer simulation of critic-based controller test performance on fuzzy beam. After several training cycles (training the critic, training the controller, retraining the critic, etc.), the controller was able to balance the ball on the fuzzy surface.
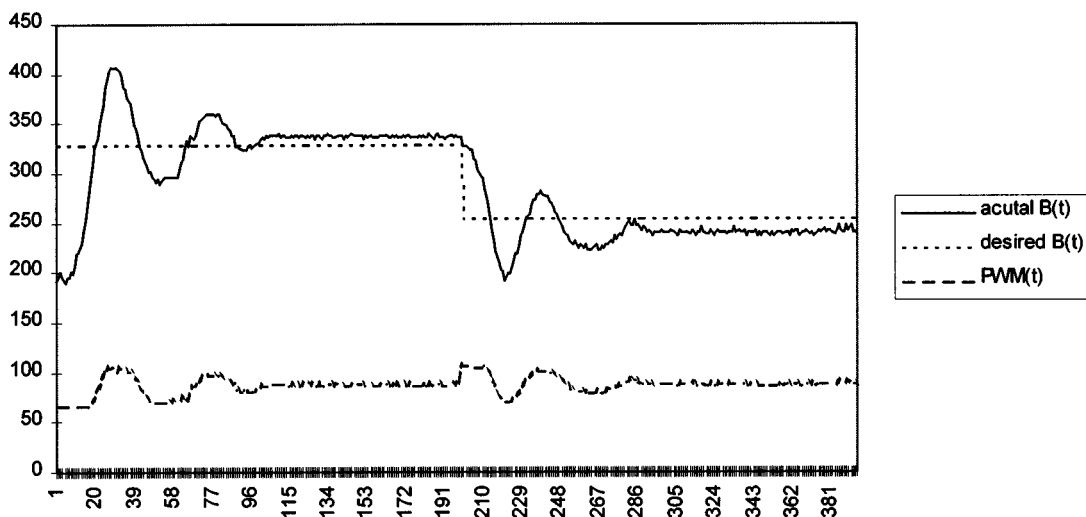

Fig. 16. Hardware results of critic-based controller test performance on fuzzy beam. The critic-based controller was able to balance the ball on the fuzzy surface.

TABLE II
COMPARATIVE RESULTS OF THE NEUROCONTROLLERS FOR THE "FUZZY" BALL-AND-BEAM

Neurocontroller	Settling time, sec	Rise time, sec	Overshoot, %
Conventional	Did not settle	1.3	37
Adaptive critic-based	6.5	1.5	31

Such discrepancies are not very surprising. After all, the sticky tape is intentionally used to complicate accurate long-term predictability of the ball movement.

Table II quantifies performance of both neurocontrollers in terms of settling time, rise time, and the overshoot. All numbers represent averages over 20 test trials of balancing the ball. Unlike the results in Table I, the settling time of the conventional controller is not reported since in most of the trials the ball either got stuck far from the desired balancing point or never stopped oscillating. We note that the steady-state error of the conventional controller, however, is not particularly large.

(The critic-based controller has some steady-state error too.) We therefore do not claim that a conventional neurocontrol solution better than ours can not be designed in fact, we suspect that it can be done.

C. Balancing in a Given Time Interval

It is interesting to change the problem so as to permit the user to specify *when* it is desirable to have the ball balanced at a particular location on the beam. To implement this, a simple modification of the proposed neurocontrollers turns out to be sufficient.

Let us consider a structure described by the following equations:

$$T(t) = T(t-1) + u(t)\Delta \tag{11}$$

$$I(t) = \begin{cases} 0, & \text{if } T(t) < \Theta \\ 1, & \text{if } T(t) \geq \Theta \end{cases} \tag{12}$$

where

$$\Theta = t_b - t_s. \tag{13}$$

Here
- $u(t)$ unit step function;
- Δ input weight equal to the sampling interval (0.055 s, as in Section II);
- t_b user-specified time (in seconds) marking the end of the balancing interval;
- t_s average settling time (see Tables I and II).

In (11) one can recognize a linear recurrent node, with its only external connection $u(t)$. In addition, (12) and (13) describe a threshold node. Both of these nodes are connected in series, with the recurrent node feeding the threshold node. At each time step t all states and inputs of both the ID network and the controller are multiplied by the output $I(t)$ of the threshold node. As a result, no balancing happens as long as $I(t)$ stays at zero. The duration of the interval when $I(t)$ remains zero (in time steps) is determined by the user specifying t_b (Δ and t_s are the predefined system parameters). The balancing interval begins at $t = 0$ with initialization of the state T to zero. Due to using the average settling time t_s the balancing may only approximately be completed by the end of the time interval from zero to t_b. In addition, if $t_b < t_s$ the balancing begins instantly (i.e., in the very first time step) but, in this case it would be impossible to meet such an invalid user specification.

Equations (11)–(13) are interpreted as a special neural network. This network is added to either of the neurocontrollers discussed in Sections VI-A and VI-B to make them immediately applicable for balancing the ball within the time specification.

VII. CONCLUSION

In this paper we have demonstrated that the control of the fuzzy ball-and-beam system, as described here, can be done without fuzzy logic, using only neural networks. It is, to our knowledge, the first reported experimental study to respond to Prof. Zadeh's challenge. Furthermore, to the best of our knowledge, this is the first accomplished implementation of DHP to control a physical system. These results showcase the capabilities of DHP in particular and neural-network controllers in general. Although we do not claim that the results presented here represent the best possible approach to this particular problem, we do believe that reinforcement learning approaches offer a valuable alternative when simulation of the system to be controlled can yield only qualitatively accurate predictions, which was the real crux of this challenge.

REFERENCES

[1] L. Zadeh, "Fuzzy control: Issues, contentions, and perspectives," in *Proc. IFAC 13th Triennial World Congr.*, San Francisco, CA, 1996, pp. 35–38.
[2] K. S. Narendra, "Adaptive control using neural networks," in *Neural Networks for Control*, W. T. Miller, R. Sutton, and P. Werbos, Eds. Cambridge, MA: MIT Press, 1990, pp. 115–143.
[3] K. S. Narendra and P. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.
[4] G. Puskorius and L. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter-trained recurrent networks," *IEEE Trans. Neural Networks*, Mar. 1994.
[5] D. R. Hush and B. Horne, "Progress in supervised neural networks—What's new since Lippman?," *IEEE Signal Processing Mag.*, pp. 8–34, Jan. 1993.
[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," in *Machine Learning*.   Boston, MA: Kluwer, vol. 3, pp. 9–44.
[7] N. Godfrey, H. Li, Y. Ji, and W. Marcy, "Real time fuzzy logic controller for balancing a beam-and-ball system," in *Fuzzy Logic and Intelligent Systems*, H. Li and M. Gupta, Eds. Norwell, MA: Kluwer, 1995, pp. 148–157.
[8] M. I. Jordan, "Supervised learning and systems with excess degrees of freedom," Dept. Comput. Sci. Inform., Univ. Massachusetts, Amherst, COINS Tech. Rep. 88-27, 1988.
[9] R. S. Sutton, "First results with dyna, an integrated architecture for learning, planning and reacting," in *Neural Networks for Control*, W. T. Miller, R. Sutton, and P. Werbos, Eds. Cambridge, MA: MIT Press, 1990, pp. 179–189.
[10] P. J. Werbos, "A menu of designs for reinforcement learning over time,," in *Neural Networks for Control*, W. T. Miller, R. Sutton, and P. Werbos, Eds. Cambridge, MA: MIT Press, 1990, pp. 67–95.
[11] ——, "Neurocontrol and supervised learning: An overview and evaluation," in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992, pp. 65–89.
[12] ——, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992, pp. 493–525.
[13] R. E. Bellman, *Dynamic Programming*.   Princeton, NJ: Princeton University Press, 1957, pp. 1–54.
[14] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560, 1990.
[15] R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, Architecture, and Applications*, Y. Chauvin and D. Rumelhart, Eds: LEA, 1995, ch. 13, pp. 311–340.
[16] G. Puskorius and L. Feldkamp, "Truncated backpropagation through time and Kalman filter training for neurocontrol," in *Proc. IEEE Int. Conf. Neural Networks (ICNN)*, Orlando, FL, 1994, pp. 2488–2493.
[17] P. Eaton, "Control of a ball-and-beam system with neural networks," M.S. thesis, Dept. Elect. Eng., Texas Tech Univ., Lubbock, TX, 1996.
[18] P. Eaton, D. Prokhorov, and D. Wunsch, "Neurocontrollers for ball-and-beam systems," in *Intelligent Engineering Systems Through Artificial Neural Networks (Proc. Conf. Artificial Neural Networks in Engineering)*, vol. 6, C. Dagli, Ed.. New York, pp. 551–557.
[19] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Networks*, vol. 8,, pp. 997–1007, 1997.
[20] Y. Jiang, C. McCorkell, and R. Zmod, "Application of neural networks for real time control of a ball-beam system," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 5, Dec. 1995, pp. 2397–2402.
[21] K. Ng and M. Trivide, "Neural integrated fuzzy controller and real-time implementation of a ball balancing beam," in *Proc. 1996 IEEE Int. Conf. Robot. Automat.*, vol. 2, Minneapolis, MN, Apr. 1996, pp. 1590–1595.

[22] D. Prokhorov, R. Santiago, and D. C. Wunsch, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, Dec. 1995.

[23] D. V. Prokhorov, "Adaptive critic designs and their applications," Ph.D. dissertation, Dept. Elect. Eng., Texas Tech Univ., Lubbock, 1997.

[24] L. Feldkamp and G. Puskorius, "A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering, and classification," *Proc. IEEE*, vol. 86, pp. 2259–2277, 1998.

Paul H. Eaton is a native of Albuquerque, NM. He received the bachelor's degree in electrical engineering from Texas Tech University, Lubbock, in May 1994. He continued at Texas Tech, working in the Applied Computational Intelligence Laboratory, on his master's thesis. He received the master's degree in electrical engineering in December 1996.

He has worked for Mission Research Corporation in Albuquerque since October 1996, designing radiation-hardened electronic components.
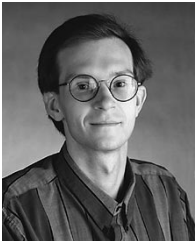
Danil V. Prokhorov (S'95–M'97) received the M.S. degree from the State Academy of Aerospace Instrument Engineering, St. Petersburg, Russia, in 1992, and the Ph.D. degree in electrical engineering from Texas Tech University, Lubbock, in 1997.

Prior to joining Texas Tech University, Lubbock, in 1994, he worked at the Institute for Informatics and Automation of Russian Academy of Sciences, St. Petersburg, as a Research Engineer. He also worked in the Vehicle Electronic Systems Department of Ford Research Laboratory, Dearborn, MI, as a Summer Intern from 1995 to 1997. He is currently with the group of Artificial Neural Systems and Fuzzy Logic, Ford Research Laboratory. His research interests include development of various methods based on artificial neural networks and their application to system modeling, control, optimization, and adaptive critics.

Dr. Prokhorov is a member of the International Neural Network Society (INNS). He is also a recipient of the 1999 INNS Young Investigator Award.

Donald C. Wunsch, II (S'86–M'90–SM'94) received the B.S. degree in applied mathematics from the University of New Mexico, Albuquerque, in 1984, the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1991 and 1987, and completed a Humanities Honors Program at Seattle University in 1981.

He was a Senior Principal Scientist at Boeing, where he invented the first optical implementation of the ART1 neural network, featured in the 1991 Boeing Annual Report, and other optical neural networks and applied research contributions until 1993. Then he was an Assistant and later Associate Professor of Electrical and Computer Engineering, and Computer Science, and Director of the Applied Computational Intelligence Laboratory at Texas Tech University, Lubbock. As of July 1999, he is the Mary K. Finley Missouri Distinguished Professor of Computer Engineering in the Department of Electrical and Computer Engineering, University of Missouri at Rolla. He has also worked for International Laser Systems and Rockwell International, and consulted for Sandia Labs, White Sands Missile Range, and Accurate Automation Corporation. Current research activities include adaptive critic designs, neural optimization, forecasting and control, financial engineering, fuzzy risk assessment for high-consequence surety, graph theory, intelligent agents, and Go. He is heavily involved in research collaborations with former Soviet scientists.

Dr. Wunsch is an Academician in the International Academy of Technological Cybernetics, and in the International Informatization Academy; and is recipient of the Halliburton Award for excellence in teaching and research at Texas Tech, and a National Science Foundation CAREER Award. He is a member of Eta kappa Nu, Sigma Xi, Phi Kappa Phi, the International Neural Network Society, Association for Computing Machinery, a life member of the American Association of Artificial Intelligence, and previously served as an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS and voting member of the IEEE Neural Network Council.