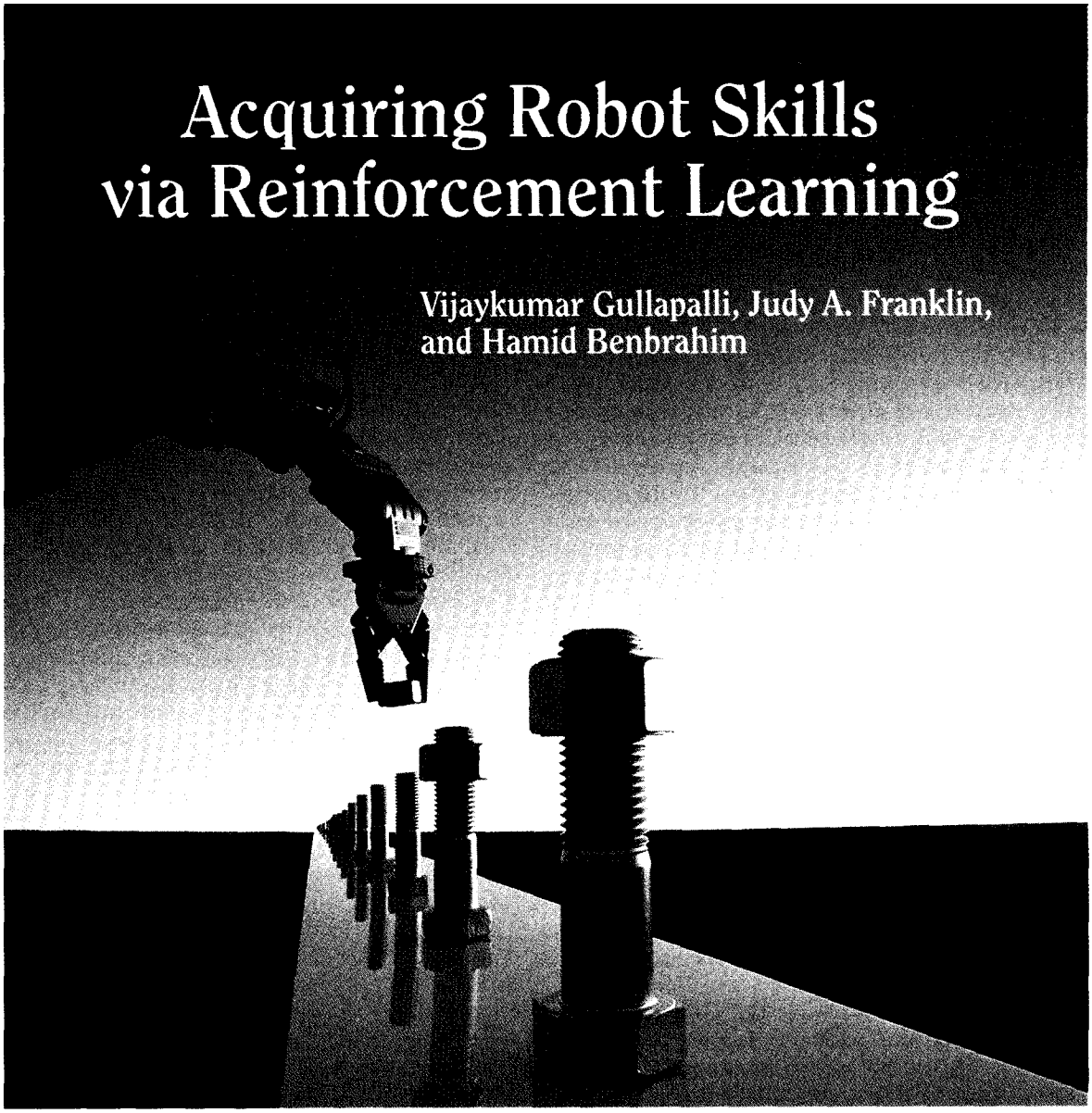
Acquiring Robot Skills via Reinforcement Learning

Vijaykumar Gullapalli, Judy A. Franklin, and Hamid Benbrahim



FPG International/Charly Franklin

Skill acquisition is a difficult, yet important problem in robot performance. In this work we focus on two skills, namely robotic assembly and balancing and on two classic tasks to develop these skills via learning: the peg-in-hole insertion task, and the ball balancing task. A stochastic real-valued (SRV) reinforcement learning algorithm is described and used for learning control, and we show how it can be used with nonlinear (multilayer) artificial neural networks. In the peg-in-hole insertion task, the SRV network successfully learns to insert a peg into a hole with extremely low clearance, in spite of high sensor noise. In the ball balancing task, the SRV network successfully learns to balance the ball with minimal feedback. Both systems are demonstrated in hardware.

V. Gullapalli is with the Computer Science Department, University of Massachusetts, Amherst, MA 01003. Email: vijaykumar@cs.umass.edu. J.A. Franklin and H. Benbrahim are with GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA 02254. Email: jfranklin@gte.com and hbenbrahim@gte.com. *The work of V. Gullapalli was supported by funding to A. Barto by the AFOSR, Bolling AFB, under Grant AFOSR-449620-93-1-0269 and by the NSF under Grant ECS-92-14866, and to R. Grupen by the NSF under Grants CDA-8922572, IRI-9116291, and IRI-9208920 (CRICCS).*

Skill Acquisition for Robots

Skilled behavior involves the effective use of knowledge in execution or performance. A skill may require dexterity or coordination, and generally develops over time through learning. This work focuses on employing learning to enable a robot to acquire skills, particularly physical skills where learning control is required. The requirements are i) dealing with nonlinearity/complex dynamics, ii) achieving robust performance under

 0272-1708/94/$04.00©1994IEEE

uncertainty, and iii) learning control actions when feedback is delayed and minimal.

Controllers must deal effectively with nonlinear robot kinematics and dynamics, complex interactions between the robot and its environment, and insufficient or inaccurate information. Unfortunately, there are many types of uncertainties and nonlinearities that traditional control design techniques do not take into account, unless a complex control system is built. Hence increasing attention is being focused on methods for learning control. We study two control problems that exemplify these issues: 1) the peg-in-hole insertion task and 2) ball balancing. Both problems involve dealing with nonlinear processes, and both are classic problems used by roboticists and control engineers to test various approaches to control.

The peg-in-hole insertion problem is difficult to model analytically and exemplifies the difficulties raised by uncertainty in real-world control problems. It is a canonical assembly task and is, in fact, the most frequent assembly operation. The ball balancing problem task is also nonlinear in nature and presents difficulties for control, requiring learning of control actions when feedback is delayed and minimal. For robotics, balancing is an important skill that can be applied to carrying objects or transferring several objects at once from arm to arm. It is also essential to walking.

We focus on a particular method for learning skilled robot control: direct real-valued reinforcement learning. This method is described in the next section. Following this we describe the SRV reinforcement learning algorithm and present some theoretical results. The fourth section contains an in-depth description of two ways of using SRV units in a nonlinear artificial neural network. The first way, with backpropagation, is used in the peg-in-hole insertion task. The second, with a boxes representation, is used in ball balancing. The final sections present details of the implementations of SRV-based controllers for the peg-in-hole insertion task and for the ball balancing task.

Reinforcement Learning and Control

Learning control involves modifying the controller's behavior to improve its performance as measured by some predefined *index of performance* (IP). If control actions that improve performance are known, supervised learning methods, or methods for learning from examples, can be used to train the controller. Unfortunately, in many control tasks, it is difficult to obtain training information in the form of prespecified control actions, in which case supervised learning methods are not directly applicable. At the same time, evaluating a controller's performance according to some IP is often fairly straightforward. In such situations, appropriate control behavior must be inferred from observations of the IP, and hence these tasks are ideally suited for the application of *associative reinforcement learning* [3].

In associative reinforcement learning, the learning system's interactions with its environment are evaluated, and the goal of the learning system is to learn to respond to (or associate) each input with the action that has the best predicted evaluation. When learning to control a plant, the learning system is the controller, its actions are control signals, and the evaluations are based on the IP associated with the control task.

In control engineering-based adaptive control, a distinction is made between direct and indirect control design methodologies (e.g., [16], [28]). Indirect methods obtain control parameters via explicit identification of a prespecified model. The model parameters are estimated on-line, and the control parameters are functions of the model parameters. Direct methods, on the other hand, estimate the controller parameters directly on-line, without constructing an explicit process model. Each method involves minimizing an index of performance and the adaptation/estimation process is based on strict stability requirements. Currently, these requirements limit the structure of the process model and/or the controller.

The distinction between direct and indirect adaptive control parallels and inspires a similar distinction in reinforcement-based learning control. Following Gullapalli [20], we distinguish between direct and indirect associative reinforcement learning methods as follows. Indirect reinforcement learning methods construct and use a model of the environment, while direct reinforcement learning methods do not. The application of indirect methods to learning control problems therefore involves two distinct operations: construction of an adequate model, which can itself be regarded as a learning problem, and using the model to train the controller. As alternatives to this, direct methods rely on perturbing the process and observing the consequences on the IP to obtain the required training information.

As we have argued previously [4], [20], handcrafting or learning an adequate model — imperative if one is to use indirect methods for training the controller — can be very difficult in some situations. This is because in these situations, either the model is too inaccurate to be useful for training the controller, or the form it is expressed in is inadequate for obtaining useful training information. For example, as we shall illustrate using the peg-in-hole insertion task, obtaining a sufficiently accurate model might be impossible in the presence of uncertainty in sensing and control. Likewise, although fairly accurate models exist for complex processes such as chemical reactions, these models often are in a form (e.g., differential equations) that provides very little useful information for training a controller for the modeled processes. Therefore, it can be expeditious to use direct reinforcement learning methods in such situations.

Noisy sensors have a two-fold effect on learning and control. First, a high noise level might seriously slow down the learning or modeling process. Second, as the controller gets wrong readings from the sensors, it can produce wrong actions. Statistical control methods can alleviate this problem, but all of them either severely increase the complexity of the controller or require some knowledge about the characteristics of the noise. Reinforcement learning methods are more efficient. They do not require an explicit feedback signal, and since learning proceeds on-line, the noise characteristics are taken into account in learning control.

Delayed evaluative feedback is another important issue in reinforcement learning. The controller may receive evaluative feedback for a control action several time steps (exact number unknown) after the action was taken. This is the case in the ball balancing problem. The use of an actor-critic reinforcement learning architecture can solve this problem and is compatible with the SRV algorithm.

In this article, peg-in-hole insertion and ball balancing tasks are used as examples to illustrate the utility of direct associative reinforcement learning methods for learning control under real-world conditions. Artificial neural networks (also known as connectionist networks) have previously been used to implement direct associative reinforcement learning controllers (e.g., Waltz

and Fu [34], Barto *et al.* [5], Franklin {12]-[14], Benbrahim *et al.* [6], Anderson [1]). Many of these implementations involved bang-bang controllers, in which the learner had only two actions to choose from. We use a fairly new reinforcement learning algorithm that makes a continuum of actions available to the controller. The particular algorithm used to train the controllers is described in the next section.

Real-Valued Reinforcement Learning

A primary requirement for learning control is the availability of a method for learning real-valued control outputs. The degrees of freedom of most robots and other controlled systems take on continuous values, and real-valued control outputs are necessary for fine motion control and smooth movement. In this section, we describe a direct reinforcement learning method for learning real-valued functions that is based on the SRV unit algorithm of Gullapalli [18].

SRV Algorithm

The SRV unit uses the Gaussian distribution to produce a stochastic output for each input x_n at time step n. Two internal parameter vectors θ_n and ϕ_n are used to compute the two parameters μ_n and σ_n of the Gaussian distribution. $\mu_n = \theta_n^T x_n$ and $\sigma_n = s(\hat{r}_n)$, where $\hat{r}_n = \phi_n^T x_n$ is the predicted evaluation given input x_n. The function $s(.)$ is a monotonically decreasing, nonnegative function of $\hat{r}_n$ with $s(1.0) = 0.0$ (assuming that the maximum expected evaluation signal is 1.0). A simple example is $s(\hat{r}_n) = \max(0.0, 1.0 - \hat{r}_n)$. The output of the unit is computed as

$$z_n = \Psi(\mu_n, \sigma_n) \tag{1}$$

where Ψ is the Gaussian distribution.

The evaluative feedback or reinforcement $r(z_n,x_n)$ from the environment is used to adjust future outputs by updating the parameter vectors θ_n and ϕ_n. The unit uses the following algorithm to update the parameter vector θ_n:

$$\theta_{n+1} = \theta_n + \alpha (r (z_n , x_n) - \hat{r}_n) \left(\frac{z_n - \mu_n}{\sigma_n} \right) x_n . \tag{2}$$

The parameter vector ϕ_n used for predicting the evaluation is updated as:

$$\phi_{n+1} = \phi_n + \beta(r (z_n, x_n) - \hat{r}_n) x_n . \tag{3}$$

In the above, α and β are learning rate parameters.

The SRV algorithm embodies the following idea. The fraction in (2) represents the *normalized perturbation* added to the mean output of the unit for the given input. If this perturbation has caused the unit to receive an evaluation signal that is *more* than the predicted evaluation, then it is desirable for the unit to produce an output closer to the actual output z_n. The mean output value should therefore be changed in the direction of the perturbation. On the other hand, if the evaluation received is *less* than the predicted evaluation, then the unit should adjust its mean in the direction *opposite* to that of the perturbation. Equation (2) above is designed to achieve this desired effect on the mean output. The mean output of the SRV unit is akin to the output of a standard neural network processing element, such as a linear or logistic unit, and the parameter vector θ is akin to the standard weight vector. However, unlike most standard units, the SRV unit is stochastic, and uses an additional parameter vector ϕ to control its stochasticity.

The stochasticity of the SRV unit serves two purposes. Randomly perturbing the mean output and observing the consequent change in the evaluation enables the unit to estimate the gradient of the evaluation with respect to its output. As described above, this enables the unit to search for and learn outputs that yield increasingly higher evaluations. In addition, by using the predicted evaluation to compute the standard deviation σ_n, the SRV unit can control the *extent* of search for the best output for each input. As the mean output μ_n for input x_n gets closer to the best possible output, the predicted evaluation $\hat{r}_n$ gets closer to the maximum possible evaluation of one. This, in turn, causes the standard deviation $\sigma_n = s(\hat{r}_n)$ to shrink, thereby restricting the search for a better output to a smaller neighborhood of the current mean output. When the mean output is optimal, $\hat{r}_n$ equals one and the standard deviation becomes zero, causing the unit's output to be the optimal mean output.

Theoretical Aspects

It is useful to rewrite (2) as

$$\theta_{n+1} = \theta_n + \alpha(r_n - \hat{r}_n)e_\theta \tag{4}$$

where α is a learning rate, $\hat{r}_n$ *is the predicted evaluation, and* e_θ is called the characteristic eligibility [36]. The predicted evaluation can be generated by a separate network, or can be the expected value of r. The characteristic eligibility is calculated from the following (also see [36]):

The probability mass function for the Gaussian distribution is

$$g = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{5}$$

and the characteristic eligibility is

$$e_\theta = \frac{\partial \ln g}{\partial \theta} = \frac{\partial \ln g}{\partial \mu} \frac{\partial \mu}{\partial \theta} = \frac{(z - \mu)}{\sigma^2} \frac{\partial \mu}{\partial \theta} = \frac{(z-\mu)}{\sigma^2} x . \tag{6}$$

If α is a nonnegative real number and $\hat{r}$ is conditionally independent of the action $z_n = \Psi(\mu_n,\sigma_n)$, given θ_n and x_n, then this algorithm belongs to a category of REINFORCE ("REward Increment = nonnegative Factor × Offset Reinforcement × Characteristic Eligibility") algorithms. In particular, if α is proportional to σ, then (4) and (2) are the same. Williams [36] devised the acronym REINFORCE and proved the following theorem.

Let $\Delta\theta = \theta_{n+1} - \theta_n$, E denote the expectation operator, and $E\{r|\theta\}$ be the performance measure that is to be maximized.

Theorem For any REINFORCE algorithm, the inner product of $E\{ \Delta\theta|\theta\}$ and $GRAD_\theta E\{r|\theta\}$ is nonnegative, and

$$E\{\Delta\theta|\theta\} = \alpha\nabla_\theta E\{r|\theta\}. \quad (7)$$

This theorem states that the average update vector in weight space (θ space) lies in the direction for which the performance measure is increasing.

Unfortunately, there is no proof of strong convergence for this family of algorithms. However, under some assumptions that are fairly standard in stochastic approximation literature (e.g., [10]) and by choosing a learning rate proportional to σ_n^2, Gullapalli [19] has proven that a modified version of the SRV algorithm causes convergence of θ_n to a predefined unique optimal value.

Use of SRV in a Nonlinear Artificial Neural Network

The SRV unit's output is computed by passing a weighted sum of its inputs through a Gaussian distribution. Therefore, by itself, the SRV unit is not sufficient to learn the complex nonlinear control functions necessary for most control problems. The standard artificial neural network method for enabling the use of such units for learning more complex functions is to create a *multilayer network*. In this case, the output of the network is the control action. Fig. 1 shows a multilayer network with a single output unit. The inputs to this unit are the outputs of a "hidden" layer of units. The hidden layer provides a nonlinear representation of the inputs (such as peg or ball position or velocity) to the output unit.

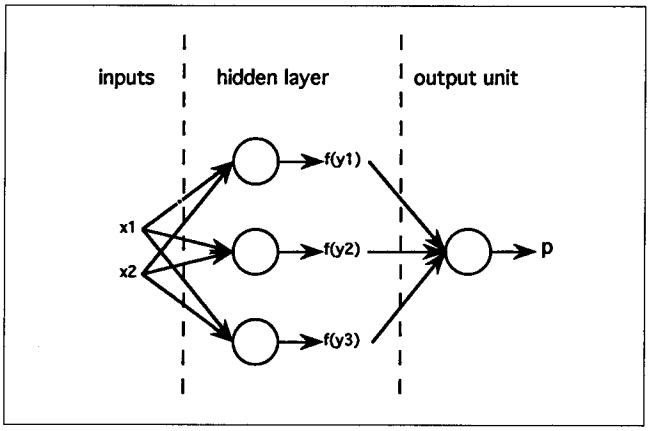


Fig. 1. A two layer artificial neural network using backpropagation.

In our implementations, we used two types of hidden layers. In one case, the hidden units compute their outputs as weighted sums of their inputs passed through a nonlinear function. Such units are popularly known as backpropagation units or multilayer perceptrons. We used this type of hidden layer in the controller for the peg-in-hole insertion task. The second type of hidden layer, called boxes, is also well-known and is a cross-product of quantized values of the inputs. The boxes hidden layer was used in the ball balancing task. Both types of hidden layers are described in this section.

Backpropagation

The backpropagation algorithm [30] described here applies to a two layer network with one output unit. The hidden layer outputs are

$$y_i = \sum_{j=1}^{n} w_{i,j}\, x_j \quad (8)$$

where $w_{i,j}$, $j = 1, \ldots n$ (n = number of inputs x_j) are the weights or parameters associated with the ith hidden unit. These weights are adapted through learning. To obtain a nonlinear network, we use a nonlinear differentiable function f, most often the sigmoid function

$$f(y_i) = \frac{1}{1+e^{-y_i}}. \quad (9)$$

The inputs to the output unit are $f(y_i)$, $i = 1, \ldots, n_h$ where n_h is the number of units in the hidden layer. The output of the network is

$$p = \Sigma_i v_i\, f(y_i) \quad (10)$$

and $\{v_i,\ i = 1, \ldots, n_h\}$ is the set of weights associated with the output unit. These weights also are adapted through learning.

A widely used technique to train this network is based on the Least Mean Squares (LMS) algorithm, which tends to minimize the square of the error. If the desired output p_d is known and p is the network's output given by (10), the error is

$$\delta = p_d - p \quad (11)$$

and we minimize

$$E = \frac{1}{2}\delta^2. \quad (12)$$

The hidden weights are updated at each time step via gradient descent as follows:

$$W_{i,j}(n+1) = w_{i,j}(n) - \alpha\frac{\partial E}{\partial w_{i,j}} \quad (13)$$

$$= w_{i,j}(n) + \alpha v_i\, f'(y_i)\delta x_j. \quad (14)$$

The term $v_i f'(y_i)\delta$ is the error backpropagated through the connection from unit i to the output unit (via weight v_i), a simple implementation of the chain rule. The weights connecting the outputs of the hidden layer to the output unit are updated by

$$v_i(n+1) = v_i(n) - \beta\frac{\partial E}{\partial v_i} \quad (15)$$

$$= v_i(n) + \beta\delta f(y_i). \quad (16)$$

In these equations, α and β are non-negative values, called the learning rates. The access of the network to exact desired outputs, and the ability to formulate the error δ, are the reasons why this form of learning is called supervised learning or learning from

examples. In our experiments, we do not have desired control outputs and therefore cannot use supervised learning. However, we can use SRV units in a backpropagation-like network in order to gain the benefits of each: nonlinearity and direct reinforcement learning. We show how to modify backpropagation for use with SRV units after first describing another type of nonlinear supervised learning network that uses *boxes*.

Boxes

Fig. 2 shows a two layer network in which the hidden layer is a set of "boxes." The range of each of the "raw" inputs x_j to the network is found and divided into equal-length segments. The cross product of all the segments along all the input dimensions defines a set of "boxes." Thus, each box demarcates a range of values along each input dimension. A binary vector of the same dimension as the number of boxes is used to represent the input value at any time step. If the input value lies in a box, the corresponding bit in the vector is set to 1, otherwise the bit is set to 0. For example, in the one-dimensional case, if the input is the value of the ball position, which has a range of -5 to 5, we might divide it into 10 segments of length 1 each. Then, when the ball position is -3.5, the boxes representation will be [0,1,0,0,0,0,0,0,0,0]. The hidden layer outputs b_i shown in Fig. 2 are the bit values of the boxes representation. Fig. 2 shows only 3 outputs but there may be many more. note that there will only be one nonzero output b_i. This is the advantage of the boxes representation. In a sense, it is a division of the input space into nonoverlapping operating modes.

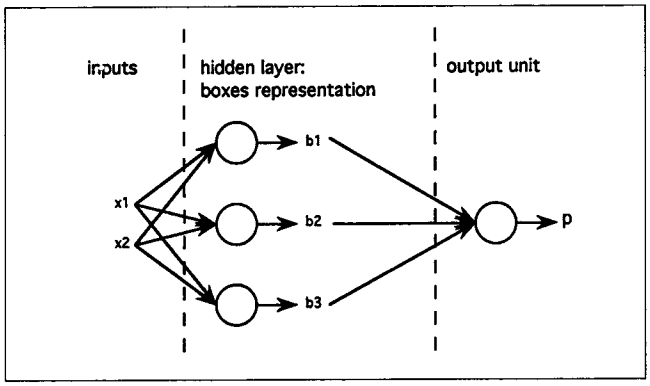


Fig. 2. A two layer artificial neural network using boxes.

The boxes representation is a fixed representation. This means that the only adaptable set of weights in the network are the output unit's weights. The output

$$p = \Sigma_i v_i b_i \ . \tag{17}$$

Note that because there is only one nonzero b_i at each increment, $p=v_j$, where $b_j = 1$, and $b_i = 0$ for all $i \neq j$. The weights are updated by

$$v_i(n+1) = v_i(n) - \beta\frac{\partial E}{\partial v_i} = v_i(n) + \beta\delta b_i \tag{18}$$

where δ is as defined in (11).

Using SRV Units in Multilayer Networks

In order to understand how SRV units can be used in a multilayer network, it is first necessary to understand the difference between error and reinforcement signals. An error signal contains information about the target (optimal) output. This gives the controller an explicit indication of the direction and the magnitude of the correction it needs to make to nullify the error. A reinforcement signal however, does not indicate on which side of the optimal output the controller's output is. Therefore, some form of search is necessary to determine the direction of the error between the current and the optimal outputs.

The SRV unit, for example, performs a local search by generating a random output. It then estimates an "error" signal

$$e_{SRV} = r - \hat{r}\left(\frac{z-\mu}{\sigma}\right)$$

from the correlation between the change in the action and the change in the reinforcement. This estimate e_{SRV} is used in place of the unknown actual error δ to update the SRV unit's parameters (see (2)). Therefore, when an SRV unit is used as an output unit in a multilayer network, it is reasonable to use the estimated error e_{SRV} in the place of δ to update the weights of the hidden layer according to (16) or (18). Fig. 3 shows a two-layer network with an SRV output unit.

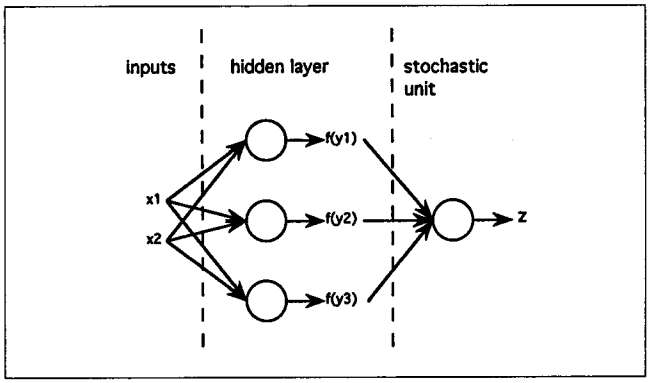


Fig. 3. SRV network.

Peg-in-Hole Insertion

Peg-in-hole insertion has been widely used by roboticists for testing various approaches to robot control. It has also been studied as a canonical robot assembly operation [17], [22], [35]. This task is also highly relevant to industrial robotics because about 33% of all automated assembly operations are peg-in-hole insertions, making them the most frequent assembly operation. The abstract peg-in-hole task can be solved quite easily (see [17] for an overview) if the exact location of the hole is known and if the manipulator can precisely control the position and orientation of the peg. However, real-world conditions of uncertainty due to 1) errors and noise in sensory feedback, 2) errors in execution of motion commands, and 3) movement of the part grasped by the robot, can substantially degrade the performance of conventional methods based on position control.

Misalignment caused by the uncertainty in positioning the peg relative to the hole can cause an insertion operation to fail. In such cases, conventional position control of the insertion opera-

examples. In our experiments, we do not have desired control outputs and therefore cannot use supervised learning. However, we can use SRV units in a backpropagation-like network in order to gain the benefits of each: nonlinearity and direct reinforcement learning. We show how to modify backpropagation for use with SRV units after first describing another type of nonlinear supervised learning network that uses *boxes*.

Boxes

Fig. 2 shows a two layer network in which the hidden layer is a set of "boxes." The range of each of the "raw" inputs x_j to the network is found and divided into equal-length segments. The cross product of all the segments along all the input dimensions defines a set of "boxes." Thus, each box demarcates a range of values along each input dimension. A binary vector of the same dimension as the number of boxes is used to represent the input value at any time step. If the input value lies in a box, the corresponding bit in the vector is set to 1, otherwise the bit is set to 0. For example, in the one-dimensional case, if the input is the value of the ball position, which has a range of -5 to 5, we might divide it into 10 segments of length 1 each. Then, when the ball position is -3.5, the boxes representation will be [0,1,0,0,0,0,0,0,0,0]. The hidden layer outputs b_i shown in Fig. 2 are the bit values of the boxes representation. Fig. 2 shows only 3 outputs but there may be many more. note that there will only be one nonzero output b_i. This is the advantage of the boxes representation. In a sense, it is a division of the input space into nonoverlapping operating modes.

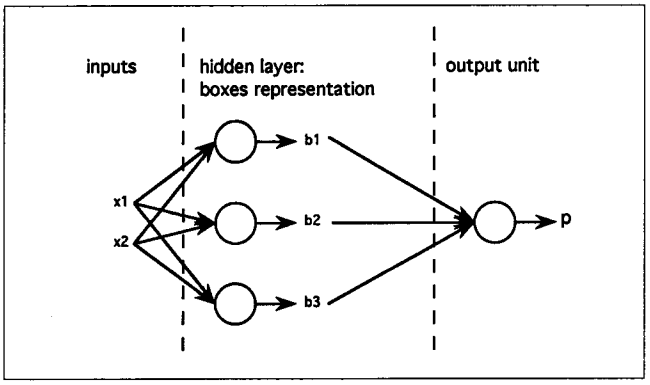


Fig. 2. A two layer artificial neural network using boxes.

The boxes representation is a fixed representation. This means that the only adaptable set of weights in the network are the output unit's weights. The output

$$p = \Sigma_i v_i b_i \ . \tag{17}$$

Note that because there is only one nonzero b_i at each increment, $p=v_j$, where $b_j = 1$, and $b_i = 0$ for all $i \neq j$. The weights are updated by

$$v_i(n+1) = v_i(n) - \beta\frac{\partial E}{\partial v_i} = v_i(n) + \beta\delta b_i \tag{18}$$

where δ is as defined in (11).

Using SRV Units in Multilayer Networks

In order to understand how SRV units can be used in a multilayer network, it is first necessary to understand the difference between error and reinforcement signals. An error signal contains information about the target (optimal) output. This gives the controller an explicit indication of the direction and the magnitude of the correction it needs to make to nullify the error. A reinforcement signal however, does not indicate on which side of the optimal output the controller's output is. Therefore, some form of search is necessary to determine the direction of the error between the current and the optimal outputs.

The SRV unit, for example, performs a local search by generating a random output. It then estimates an "error" signal

$$e_{SRV} = r - \hat{r}\left(\frac{z-\mu}{\sigma}\right)$$

from the correlation between the change in the action and the change in the reinforcement. This estimate e_{SRV} is used in place of the unknown actual error δ to update the SRV unit's parameters (see (2)). Therefore, when an SRV unit is used as an output unit in a multilayer network, it is reasonable to use the estimated error e_{SRV} in the place of δ to update the weights of the hidden layer according to (16) or (18). Fig. 3 shows a two-layer network with an SRV output unit.

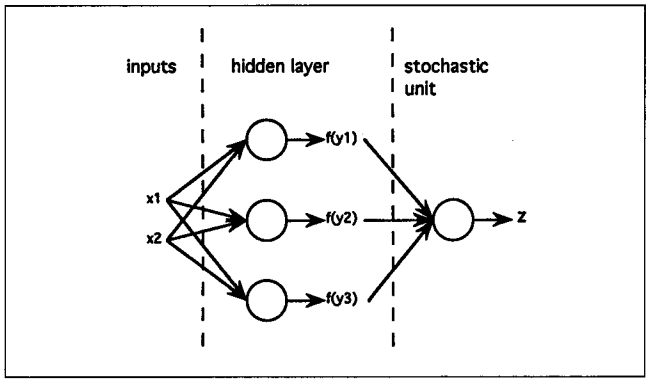


Fig. 3. SRV network.

Peg-in-Hole Insertion

Peg-in-hole insertion has been widely used by roboticists for testing various approaches to robot control. It has also been studied as a canonical robot assembly operation [17], [22], [35]. This task is also highly relevant to industrial robotics because about 33% of all automated assembly operations are peg-in-hole insertions, making them the most frequent assembly operation. The abstract peg-in-hole task can be solved quite easily (see [17] for an overview) if the exact location of the hole is known and if the manipulator can precisely control the position and orientation of the peg. However, real-world conditions of uncertainty due to 1) errors and noise in sensory feedback, 2) errors in execution of motion commands, and 3) movement of the part grasped by the robot, can substantially degrade the performance of conventional methods based on position control.

Misalignment caused by the uncertainty in positioning the peg relative to the hole can cause an insertion operation to fail. In such cases, conventional position control of the insertion opera-

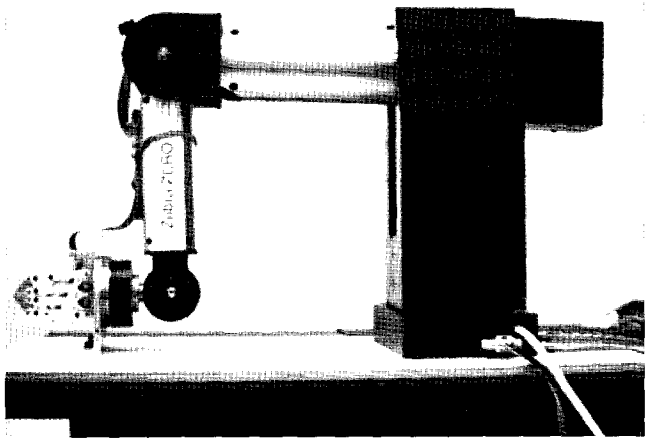


Fig. 4. The Zebra Zero robot used for the peg-in-hole insertion task.

tion has to be augmented with some method of overcoming possible misalignments. One approach is to use geometric path planning techniques to determine a motion sequence that has the highest likelihood of success given the peg and hole geometries and the margins of uncertainty in executing that sequence. *Preimaging* [26], *backprojection* [11], and other strategies (e.g., [7], [9]) have been proposed for geometric path planning.

Alternative approaches, based on reactive control, try to counter the effects of uncertainty with on-line modification of the insertion path based on sensory feedback. Compliant motion control, in which the motion path is modified by contact forces or tactile stimuli occurring during the motion, is often used. We focus on force-guided insertion strategies that cause the manipulator to deviate from its nominal motion path depending on the sensed contact forces. Force control serves two purposes. First, it can be used to prevent excessive contact forces from damaging the peg and hole and the manipulator during assembly. More importantly, force control can correct misalignment by mapping contact forces arising from misalignment to motions that reduce the misalignment.

The key component in implementing force-guided assembly is the difficult task of specifying the manipulator's admittance, which determines how the manipulator responds to forces. A common approach (e.g., [31]), sometimes referred to as the two phase approach, involves first determining a nominal collision-free assembly path assuming there are no uncertainties in positioning the parts. In the second phase, this path is analyzed to determine what kinds of collisions are likely to occur at any point along the path given a model of the uncertainty. The contact forces are computed at the points of collision using a model of the physics of interaction of rigid bodies, and a corresponding corrective motion is determined. An admittance mapping is then obtained by interpolating a suitable function over this set of contact force–corrective motion pairs.

There are practical limitations in implementing this two phase approach. First, because only a limited number of data points is used to determine the admittance mapping, one cannot be sure that all possible contact forces will be mapped to corrective motions. Moreover, determining which manipulator configuration and contact force combinations are critical in forming the admittance mapping is a hard problem. As Asada [2] points out, many assembly tasks require complex nonlinear admittance mappings, but humans find it quite difficult to prespecify appropriate admittance behavior [26], especially in the presence of uncertainty and noise. A second problem is the approach's reliance on models of both the uncertainty and the interaction forces between parts in designing the admittance. Both are notoriously difficult to model in the presence of friction and when the robot is interacting physically with its environment. Therefore, the synthesized admittance might not perform as intended in practice.

Because of the shortcomings of these approaches, peg-in-hole insertion under uncertainty is a good candidate problem for applying learning approaches (e.g., [18], [32]). This is demonstrated here by using a Zebra Zero robot shown in Fig. 4 to perform peg-in-hole insertions. This robot is equipped with a wrist force sensor, and can also sense positions of the joints of the arm through position encoders.

We performed experiments to quantify the uncertainty of these sensors. In order to quantify the position uncertainty during interactions between the peg and the hole, we compared the sensed peg position with its actual position in Cartesian space when different forces were acting on the peg. The robot was commanded to maintain a fixed position under five different load conditions applied sequentially: no load, and a fixed load of 0.12 kgf applied in the $\pm x$ and $\pm y$ directions. Under each condition, the position and force feedback from the robot sensors, as well as the actual x-y position of the peg were recorded.

Sensed x-y positions were computed from the joint positions sensed by the Zero's joint position encoders. We found a large discrepancy between the sensed and actual positions of the peg: while the actual change in the peg's position under the external load was of the order of 2 to 3 mm, the largest sensed change in position was less than 0.025 mm. In comparison, the clearance between the peg and the hole was 0.175 mm. Fig. 5 shows 30 time-step samples of the force sensor output for each of the load

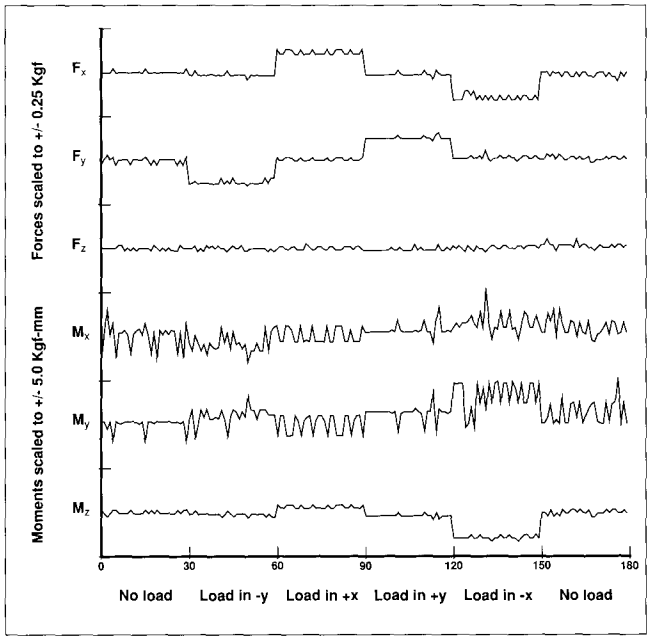


Fig. 5. Time-step samples of the sensed forces and moments under 5 different load conditions. With an ideal sensor, the readings would be constant in each 30 time-step interval.

conditions described above. As can be seen from the figure, there is considerable sensor noise, especially in recording moments.

Learning Peg-in-Hole Insertion

Our approach to learning a reactive control strategy for peg insertion under uncertainty is based on active generation of compliant behavior using a nonlinear mapping from sensed positions and forces to position commands (see also [21]). The controller learns this mapping through repeated attempts at peg insertion. It should be emphasized here that because of the uncertainty, using a PD position controller to directly servo the peg into the hole (given the nominal location of the hole) does not work, especially when the clearances are much smaller than the positional uncertainty, as is the case here. Moreover, there is no guarantee that the interaction forces will be safely bounded when using a PD position controller. Therefore, the admittance mapping learned by the network is essential for safe, reliable insertion.

Peg Insertion Task. The peg insertion task is depicted in Fig. 6. Starting from a random initial position and orientation, the robot has to move the peg in a trajectory that will result in the peg being inserted in the hole. For this task, the peg is 30 mm long and 6 mm in diameter, while the hole is *chamferless* and 6.35 mm in diameter. Thus the clearance between the peg and the hole is 0.175 mm.

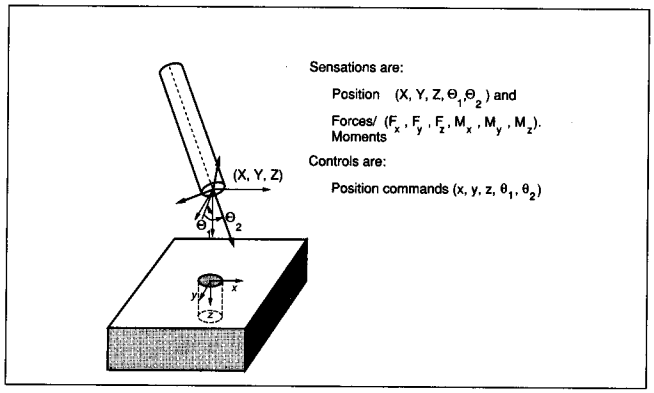


Fig. 6. The peg-in-hole insertion task.

Controller. The controller is implemented as a connectionist network that operates in closed loop with the robot. The network has 11 inputs. These are the sensed positions and forces, (X, Y, Z, Θ_1, Θ_2) and (F_x, F_y, F_z, M_x, M_y, M_z). The network's five outputs form the position command (x, y, z, θ_1, θ_2). Following Asada [2], we used two hidden layers of 30 units each. Both layers of hidden units are comprised of backpropagation units, while the output units are the SRV reinforcement learning units [18]. The extension of backpropagation from two to three layers, and from one to five output units, is a straightforward application of the chain rule.

The Cartesian position inputs to the network are computed from the sensed joint positions using the forward kinematics equations for the Zero robot. The force and moment inputs are those sensed by the six-axis force sensor. A PD servo loop servos the robot to the position output by the network at each time step, resulting in some motion of the peg.

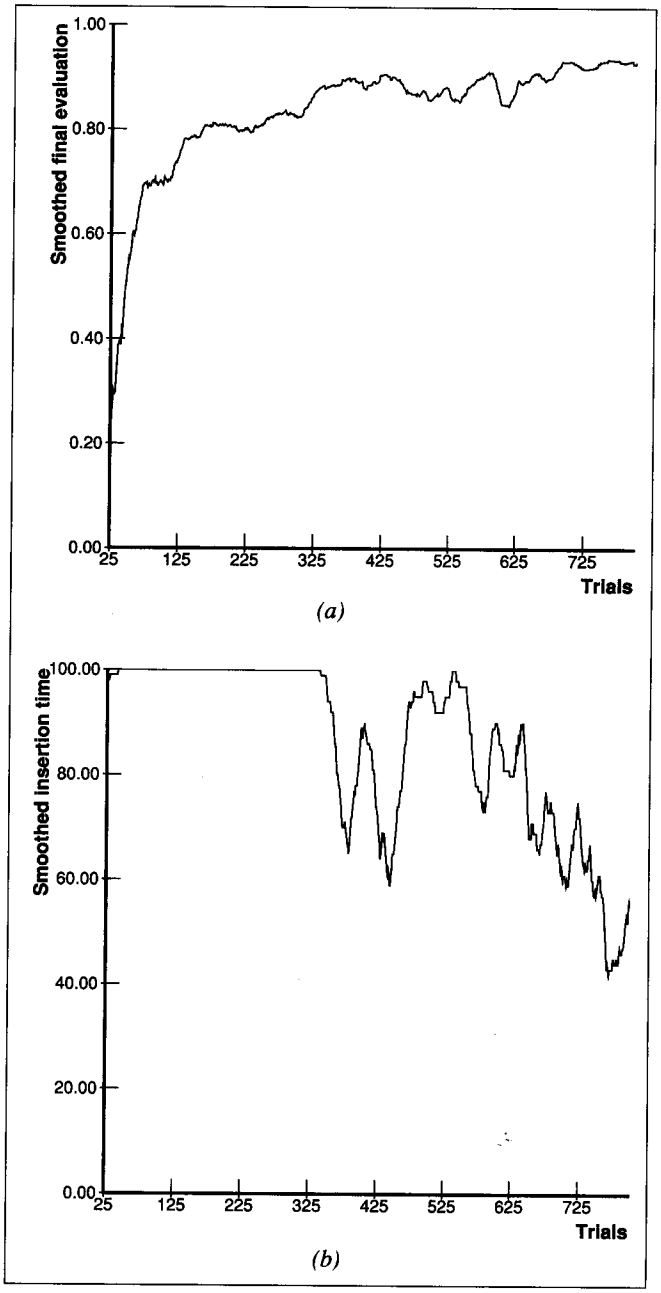


Fig. 7. Smoothed final evaluation received and smoothed insertion time (in simulation time steps) taken on each of 800 consecutive trials on the peg insertion task. The smoothed curve was obtained by filtering the raw data using a moving-average window of 25 consecutive values.

Training Methodology. The controller network is trained in a sequence of trials, each of which starts with the peg at a random position and orientation with respect to the hole and ends either when the peg is successfully inserted in the hole, or when 100 time steps have elapsed. An insertion is termed successful when the peg is inserted to a depth of 25 mm into the hole. An evaluation of the controller's performance r ranging from 0 to 1 with 1 denoting the best possible evaluation, is computed based on the new peg position and the forces acting on the peg as

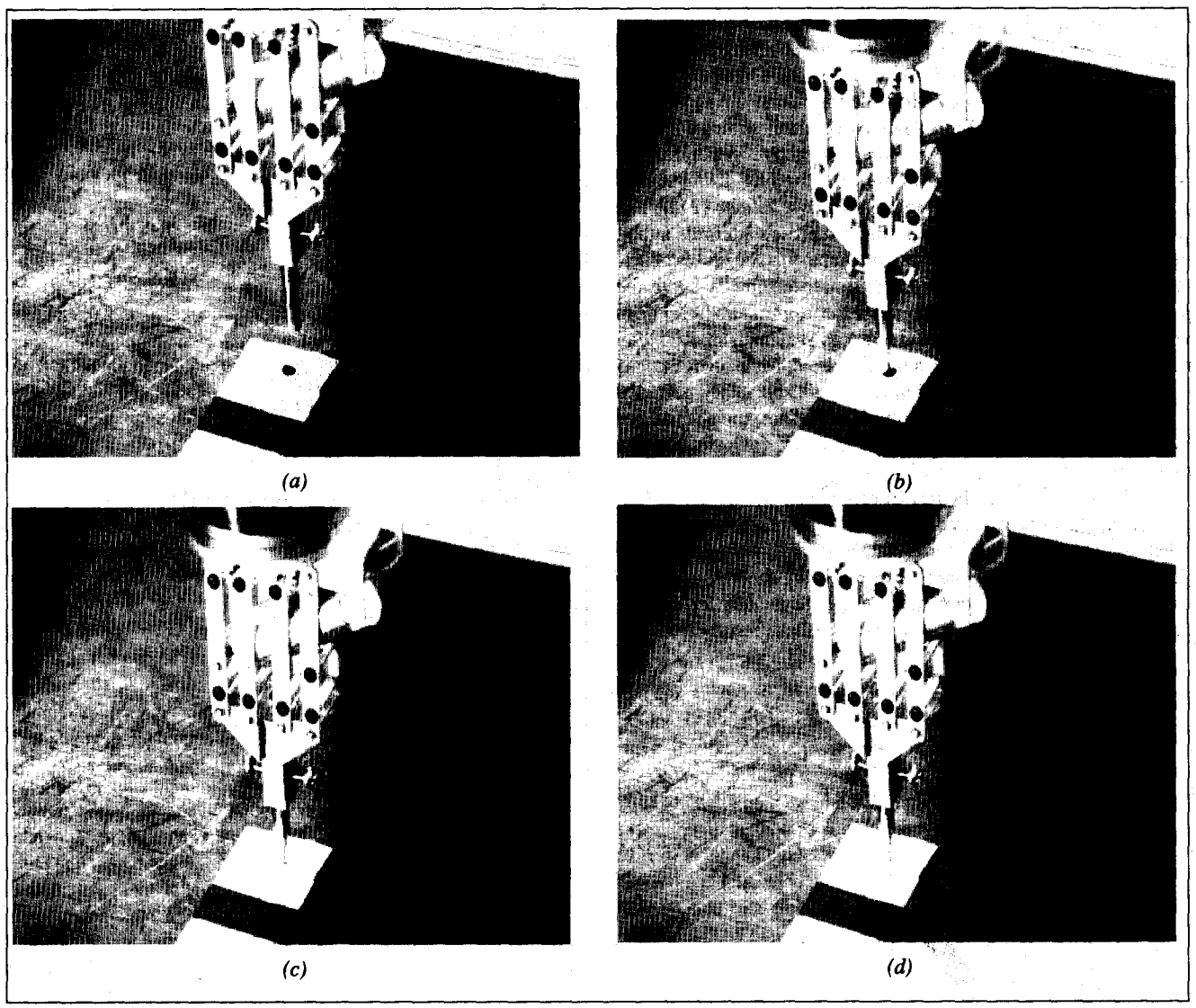


Fig. 8. Sequential images of the robot inserting the peg under the control of the trained network.

$$r = \begin{cases} \max(0.0, 1.0 - 0.01\,\|\text{position error}\|) \\ \qquad \text{if all forces} \le 0.5 \text{ Kgf} \\ \max(0.0, 1.0 - 0.01\,\|\text{position error}\|) - 0.1 F_{\max} \\ \qquad \text{otherwise} \end{cases}$$

where $F_{\max}$ denotes the largest magnitude force component.

Thus, the closer the sensed peg position is to the desired position with the peg inserted in the hole, the higher the evaluation. Large sensed interaction forces, however, reduce the evaluation, since these forces should be minimized. Using this evaluation, the network adjusts its weights appropriately and the cycle is repeated. Note that because the position error is determined from the sensed position of the peg and the nominal location of the hole, both of which are subject to uncertainty, the evaluation can be inaccurate and noisy. However, our results indicate that despite the noisy evaluation, our approach can be used to successfully train a network to perform skilled insertion.

Performance Results

A learning curve showing the final evaluation over 800 consecutive trials on the insertion task is shown in Fig. 7(a). The final evaluation levels off close to 1 after about 400 trials because after that amount of training, the controller is consistently able to perform successful insertions within 100 time steps. However, performance as measured by insertion time continues to improve, as is indicated by the learning curve in Fig. 7(b), which shows the time to insertion decreasing over the 800 trials. These curves indicate that the controller becomes progressively more *skillful* at peg insertion with training. Similar results obtained for the 2D peg insertion task are reported in [20].

Fig. 8 shows a sequence of snapshots of the robot performing an insertion under the control of the trained network. As seen in the second snapshot in the sequence, during the insertion process the peg often contacts the flat surface surrounding the hole due to the positional uncertainty. Because of the absence of a chamfer, this kind of positional uncertainty would cause difficulties for

traditional approaches. However, our controller learns a strategy for sliding the peg over the flat surface into the hole, thereby ensuring successful insertion.

Discussion

The high degree of uncertainty in the sensory feedback from the Zebra Zero, coupled with the fine motion control requirements of peg-in-hole insertion make the task under consideration an example of learning control under extreme uncertainty. The positional uncertainty, in particular, is of the order of 10 to 50 times the clearance between the peg and the hole and is primarily due to gear backlash. There is also significant uncertainty in the sensed forces and moments due to sensor noise. Our results indicate that direct reinforcement learning can be used to learn a reactive control strategy that works robustly even in the presence of a high degree of uncertainty.

Although others have studied similar tasks, in most other work on learning peg-in-hole insertion (e.g., [25]) it is assumed that the positional uncertainty is about an order of magnitude *less* than the clearance. Moreover, results are often presented using simulated peg-hole systems. Our results indicate that our approach works well with a physical system, despite the much higher magnitudes of noise and consequently greater degree of uncertainty inherent in dealing with physical systems. Furthermore, the success of the direct reinforcement learning approach to training the controller indicates that this approach can be useful for automatically synthesizing robot control strategies that satisfy constraints encoded in the performance evaluations.

Ball Balancer

The ball balancer is also a nonlinear system that is difficult to control and is a classic problem used by control theorists to study nonlinear control [23], [24]. It also requires the learning of control actions when feedback is delayed (under our assumptions for the problem). Of late it has been used as a testbed both in control education courses and in academic research labs [8]. For robotics, balancing is an important skill that can be applied to carrying objects or transferring several objects at once from arm to arm. Balancing is also essential to walking [15], [29].

The dynamic model of the ball balancer is nonlinear, even when the "jumping ball" phenomenon is disregarded [24]. Modeling of the ball balancer is not as difficult as modeling the peg-insertion dynamics, and control techniques exist that use the model and control the balancer successfully.

However, the actual system can suffer from sensor-based uncertainty, particularly when the ball jumps. For this reason, we do not rely on the position feedback as much as a traditional controller might. Instead of a position error signal, we can rely on a reinforcement signal. In fact we assume a minimal and delayed reinforcement. Also, the balancer becomes more difficult to model and control as it is generalized into a more articulated robotic system. Thus it is a good candidate for direct reinforcement learning.

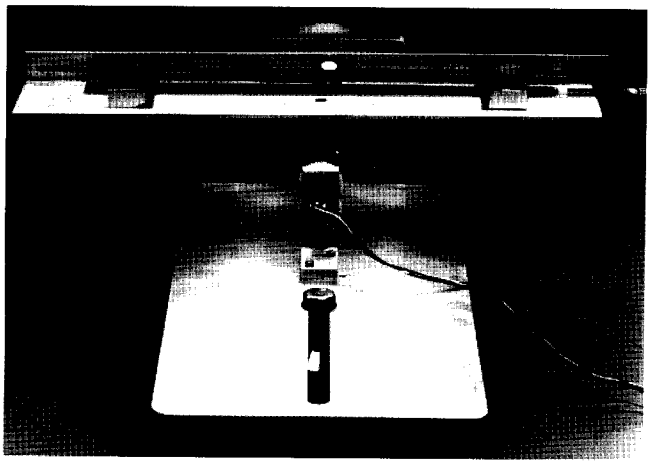


Fig. 9. The ball balancer.

Learning Ball Balancing

Our approach to ball balancing is based on work by Barto *et al.* [5] in using reinforcement learning to balance an inverted pendulum. Their work involves using a two-action reinforcement learning algorithm. We applied the two-action algorithm successfully to the ball balancer prior to applying the real-valued SRV algorithm [6]. The actions in both cases are voltages applied to the motor that moves the beam.

Ball Balancing Task. As shown in Fig. 9, the balancer is a beam made of a 16" x 2" section of aluminum attached at its center to a shaft, which a dc motor can turn in both directions. Two bumpers under the beam limit its movement to angles of about 20° from the horizontal. The beam has a one inch high fence along one side. A metal ball rolls along the fence on the beam. Bumpers at the ends of the beam prevent the ball from falling off. A pressure sensor measures the ball's position; a potentiometer attached to the axle of the beam measures the beam's angle.

The state of the balancer is represented by four variables: the position of the ball, its velocity, the angle of the beam, and its angular velocity. Both velocities are calculated by evaluating the differences of the positions at successive increments of time. This is a real-time task; at every increment (20 times a second or increments of 50 ms) the computer reads the state of the system, issues the action and updates the learning weights. The reinforcement it receives in order to update the weights is minimal. We show how the learning controller uses this minimal information next.

Controller. In the ball balancer experiment, we use Barto *et al.*'s actor-critic configuration [5] for reinforcement learning. The actor is the nonlinear artificial neural network with an SRV output unit that outputs the control action that becomes the motor voltage. The hidden layer is a set of boxes that represents the quantized state. In the ball balancing application there are 5 ball

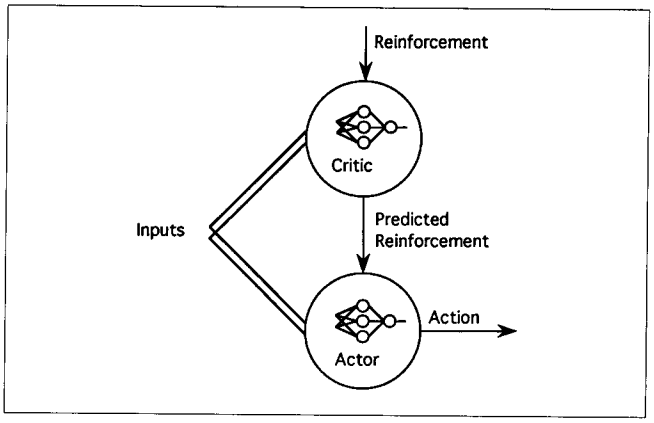


Fig. 10. The actor-critic reinforcement learning architecture.

position segments, 6 ball velocity segments, 3 angular position segments, and 2 angular velocity segments, for a total of $5 \times 6 \times 3 \times 2 = 180$ boxes (or hidden units). The hidden layer becomes the set of inputs used by the SRV.

We assume that the evaluative feedback is minimal:

$$r = \begin{cases} 0 & \text{if the ball is balanced,} \\ -1 & \text{if the ball hits either beam end.} \end{cases} \quad (19)$$

The actor-critic architecture deals effectively with minimal reinforcement. As a result the system learns that having the ball in the center of the beam with a very small speed is the safest situation. Without the critic, this situation would look to the system like any nonfailure situation.

As described in Fig. 10, the actor outputs the action or control, and the critic evaluates the action, given the "raw" performance evaluation r. The role of the critic is to assign an evaluation to each action. Without the critic, the system would receive only a 0 or -1. Since the system gets a nonzero reinforcement only when the ball fails, it needs the critic to build an evaluation function for the states that are far away from the failure states.

We rewrite the SRV weight update equation here for convenience:

$$\theta_{n+1} = \theta_n + \alpha(r(z_n, x_n) - \hat{r}_n) \left(\frac{z_n - \mu_n}{\sigma_n} \right) x_n . \quad (20)$$

The critic network outputs a prediction $q(n)$ of the reinforcement [33]. When using a critic, the predicted reinforcement $\hat{r}_n$ becomes a difference in successive predictions of reinforcement:

$$\hat{r}_n = (q(n-1) - \gamma q(n)) \quad (21)$$

where $0<\gamma<1$ is a constant gain. The idea is that the critic predicts discounted future reward. $\hat{r}_n$ is a difference between the prediction at time step n-1 and the "actual" discounted reward at step n. The critic has to use its own prediction rather than the actual r, until r is nonzero. Both the critic and SRV networks use $r-\hat{r}$ as the "error" in (20) to update their weights. Fig. 11 shows the critic's modification of the reinforcement signal.

The critic solves the problem of "delayed" reinforcement via computing a difference in successive predictions and via another mechanism called eligibility traces. When the ball fails it is not only the last action that is responsible, but a certain succession of actions. We assume that actions are more and more responsible for failure the closer they are to it in time. Eligibility traces are moving average filters that replace the eligibility e_{θ_i} of (4). Eligibility traces provide a decaying history that aids the algorithm in learning, when the reward signal is delayed.

Simply, (20) is replaced by

$$\theta_{n+1} = \theta_n + \alpha (r(z_n, x_n) - \hat{r}_n) T_\theta \quad (22)$$

where

$$\begin{aligned} T_\theta(n+1) &= \lambda T_\theta(n) + (1-\lambda) e_\theta \\ &= \lambda T_\theta(n) + (1-\lambda) \left(\frac{z_n - \mu_n}{\sigma_n} \right) x_n \end{aligned} \quad (23)$$

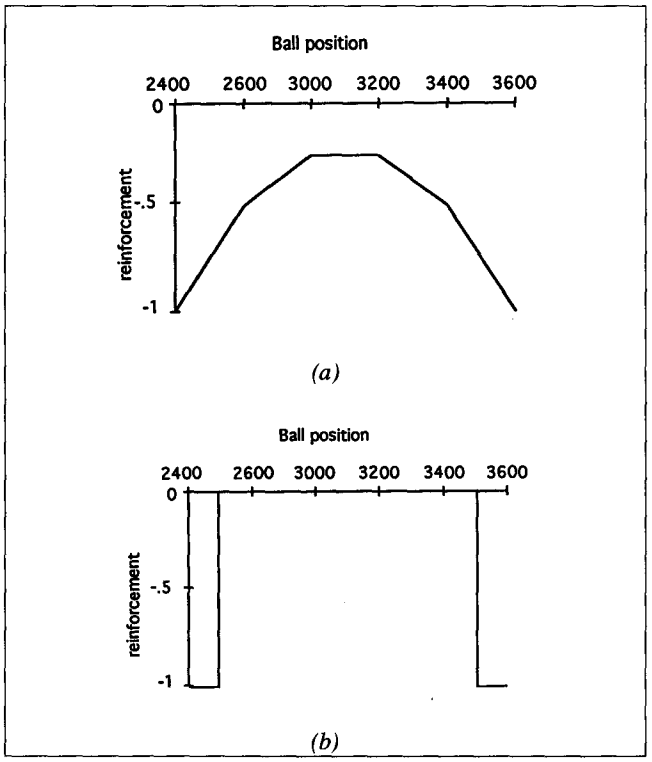


Fig. 11. (a) The modification of the reinforcement by the critic. (b) The "raw" reinforcement.

where x is the input and $0<\lambda<1$ is the decay rate. This amount of feedback, along with the use of the critic network and the eligibility factors is enough for the system to learn to control the balancer. The roots of using the critic's successive predictions plus eligibility are in classical conditioning [5], [33].

Training Methodology. The SRV unit of the controller computes an action based on the sensed positions and the computed velocities. The action is multiplied by a gain and sent to the power amp and then to the motor actuating the beam. The state is sampled at the next time increment and the reinforcement as well as the predicted reinforcement are computed. These values plus the last action taken are used to update the weights, on-line. The new weights are used to compute the next action.

When a failure occurs (the ball hits either end of the beam) the number of steps until failure is recorded and the ball is moved out of the failure zone by exerting a torque causing the beam to move toward the opposite failure zone. The number of steps until failure can be plotted as a learning curve.

Performance Results. The stochastic real-valued output-based controller successfully learns to balance the ball and improves over the two-action controller previously implemented. We show three plots in the following figures. Fig. 12 shows a plot of the number of successful steps (time increments) before each failure. The learning curve shows that after 700 failures the ball is balanced with no further failure.

Fig. 13 shows both the ball position and velocity, after the controller has learned to balance, when the ball is placed in a failure zone. Note that the position and velocity never settle

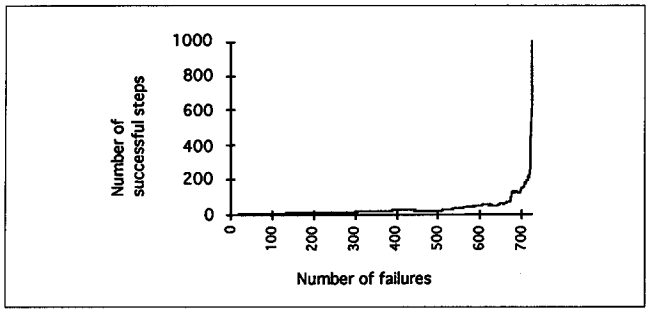


Fig. 12. Number of steps as a function of number of failures.

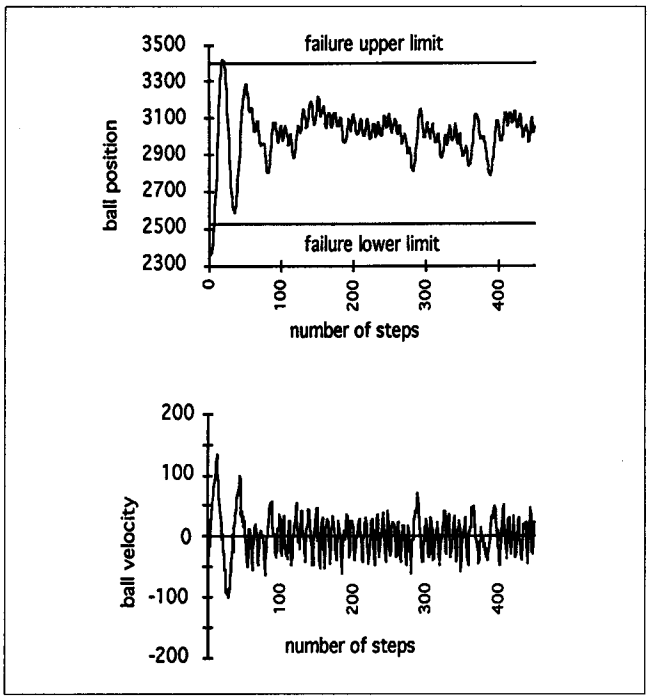


Fig. 13. Ball position and velocity as a function of steps when the ball is placed in a failure zone at the end of the beam.

down. This is a ramification of using the discrete resolution of the boxes representation.

Discussion

In our experiment the goal is to keep the ball from hitting either end. The balancer learns to keep the ball as far as possible from the failure states. One could modify this goal to make the ball go to different positions. One could add other constraints such as penalizing the balancer for actions high in magnitude, penalizing high speed, or penalizing high rates of change in successive actions. Combining these constraints together, one can obtain smooth and slow balancing.

We have tried to minimize the energy used by making r a function of $|z|$, the absolute value of the control action. The result is that the motion of the system is smoother, but the position oscillates more. When the action magnitude is small, the ball goes far from the center and then a larger control action brings it back to the center.

Clearly different performance criteria require different reinforcement functions. This is an area we are interested in pursuing.

Robots Acquiring Skills

It is important to the field of learning control and robotics to have real robots learn to acquire skills, and this continues to be our goal. Towards this end, we presented the stochastic real-valued reinforcement learning algorithm and argued for its utility in learning control. Since we are addressing nonlinear control problems, we described how the SRV unit can be used in two different multilayer artificial neural networks, both of which are capable of learning nonlinear control functions. In the peg-in-hole insertion task, the SRV was coupled with backpropagation to form a three-layer five-output network that successfully learned to skillfully insert a peg into a chamferless hole with low clearance under a high degree of uncertainty. In the ball balancing task, the SRV unit was coupled with a boxes representation and a reinforcement critic, and successfully learned to balance the ball. Both systems were real, not simulations.

The capabilities of learning control approaches to controlling real robots can be vastly extended if we use available control tools and combine them in new ways. In the past, we have combined conventional pole placement-based control with reinforcement learning [12]-[14] on simple robotic simulations. As the controlled robots become more complicated, we envision more complex control architectures that combine control modules based on SRV type learning algorithms with control modules designed using conventional control techniques. For example, a controller may learn to choose between different control modules based on their performance under different operating conditions, or the outputs of a learning system and a conventional controller may be combined to form the control signal to a nonlinear plant.

Acknowledgment

J. Franklin and H. Benbrahim would like to acknowledge the support of John Vittal for this work and the encouragement of Oliver Selfridge.

References

[1] C.W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proc. Fourth Int. Workshop Machine Learning*, Irvine, CA. 1987.

[2] H. Asada, "Teaching and learning of compliance using neural nets: Representation and generation of nonlinear compliance," in *Proc. 1990 IEEE Int. Conf. Robot. Auto.*, 1990, pp. 1237-1244.

[3] A.G. Barto and P. Anandan, "Pattern recognizing stochastic learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 360-375, 1985.

[4] A.G. Barto and V. Gullapalli, "Neural networks and adaptive control," in *Natural and Artificial Intelligence* (Research Notes in Neural Computation), P. Rudomin, M.A. Arbib, and F. Cervantes-Perez, Eds. Springer-Verlag, 1992.

[5] A.G. Barto, R.S. Sutton, and C.W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 13, pp. 835-846, 1983.

[6] H. Benbrahim, J. Doleac, J. Franklin, and O. Selfridge, "Real-Time Learning: A Ball on a Beam," in *Proc. 1992 Int. Joint Conf. Neural Networks*, Baltimore, MD, June 1992.

[7] M.E. Caine, T. Lozano-Pérez, and W.P. Seering, "Assembly strategies for chamferless parts," in *Proc. IEEE Int. Conf. Robot. Auto.*, May 1989, pp. 472-477.

[8] *Control Syst. Mag.*, vol. 12, June 1992.

[9] B.R. Donald, "Robot motion planning with uncertainty in the geometric models of the robot and environment: A formal framework for error detection and recovery," in *Proc. IEEE Int. Conf. Robot. Auto.*, pp. 1588-1593, 1986.

[10] A. Dvoretzky, "On stochastic approximation," in *Proc. Third Berkeley Symp. Math. Stat. Probability*, vol. 1. Berkeley and Los Angeles, CA: Univ. of California Press., 1956, pp. 39-55.

[11] M. Erdmann, "Using backprojections for fine motion planning with uncertainty," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 19-45, 1986.

[12] J.A. Franklin, "Learning control in a robotic system," in *Proc. 1987 IEEE Int. Conf. Syst., Man, Cybern.*, Alexandria, VA, 1987.

[13] J.A. Franklin, "Refinement of robot motor skills through reinforcement learning," in *Proc. 27th IEEE Conf. Decision Control*, Austin, TX, Dec. 1988.

[14] J.A. Franklin, "Input representation for refinement learning control," in *Proc. 4th Int. IEEE Symp. Intell. Control*, Albany, NY, Sept. 1989.

[15] J. Furusho, and A. Sano, "Sensor-based control of a nine-link biped," in *Robot. Res.*, vol. 9, pp. 83-98, Apr. 1990.

[16] G.C. Goodwin and K.S. Sin, *Adaptive Filtering, Prediction, and Control.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

[17] S.J. Gordon, "Automated assembly using feature localization," Ph.D. thesis, Tech. Rep. 932, Massachusetts Inst. of Technology, M.I.T. AI Laboratory, Cambridge, MA, 1986.

[18] V. Gullapalli "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Net.*, vol. 3, pp. 671-692, 1990.

[19] V. Gullapalli, "Associative reinforcement learning of real-valued functions," in *Proc. 1991 IEEE Int. Conf. Syst., Man, Cybern.*, Charlottesville, VA, Oct. 1991.

[20] V. Gullapalli, "Reinforcement learning and its application to control," Ph.D. thesis, Univ. Massachusetts, Amherst, MA, 1992.

[21] V. Gullapalli, "Learning control under extreme uncertainty," in *Advances in Neural Information Processing Systems*, C.L. Giles, S.J. Hanson, and J.D. Cowan, Eds. San Mateo, CA: Morgan Kaufmann, 1993.

[22] R.E. Gustavson, "A theory for the three-dimensional mating of chamfered cylindrical parts," *J. Mechanisms, Transmissions, Auto. Des.*, Dec. 1984.

[23] R.R. Kadiyala, "A tool box for approximate linearization of nonlinear systems," *Control Syst. Mag.*, vol. 13, pp. 47-57, Apr. 1993.

[24] P.V. Kokotovic, "The joy of feedback: Nonlinear and adaptive," *IEEE Control Syst. Mag.*, vol. 12, pp. 7-17, June 1992.

[25] S. Lee and M.H. Kim, "Learning expert systems for robot fine motion control," in *Proc. 1988 IEEE Int. Symp. Intell. Control*, H.E. Stephanou, A. Meystal, and J.Y.S. Luh, Eds. Washington, DC: IEEE Computer Soc., 1989, pp. 534-544.

[26] T.Lozano-Pérez, M.T. Mason, and R.H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *Int. J. Robot. Res.*, vol. 3, no. 1, pp. 3-24, Spr. 1984.

[27] D. Michie and R. Chambers, "BOXES: An experiment in adaptive control," in *Machine Intelligence*, E. Dale and D. Michie, Eds. Edinburgh, U.K.: Oliver and Boyd, 1968.

[28] K.S. Narendra, "Adaptive control using neural networks," in *Neural Networks for Control*, T. Miller, R.S. Sutton, and P.J. Werbos, Eds. Cambridge, MA: M.I.T. Press, 1990, ch. 5.

[29] M.H. Raibert, *Legged Robots That Balance.* Cambridge, MA: M.I.T. Press, 1986.

[30] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D.E. Rumelhart and J.L. McClelland, Eds., vol. 1, Foundations. Cambridge, MA: Bradford Books/M.I.T. Press, 1986.

[31] J.M. Schimmels and M.A. Peshkin, "Admittance matrix design for force-guided assembly," *IEEE Trans. Robot. Auto.*, vol. 8, no. 2, pp. 213-227, Apr. 1992.

[32] J. Simons, H.V. Brussel, J.D. Schutter, and J.Verhaert, "A self-learning automaton with variable resolution for high precision assembly by industrial robots," *IEEE Trans. Auto. Control*, vol. 27, no. 5, pp. 1109-1113, Oct. 1982.

[33] R.S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. diss., Dept. Computer and Info. Science, Univ. of Massachusetts, Amherst, MA, 1984.

[34] M.D. Waltz and K.S. Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Trans. Auto. Control*, vol. 10, pp. 390-398, 1965.

[35] D.E. Whitney, "Quasi-static assembly of compliantly supported rigid parts," *J. Dynamic Syst., Meas., Control*, vol. 104, Mar. 1982. Also in *Robot Motion: Planning and Control*, M. Brady *et al.*, Eds. Cambridge, MA: M.I.T. Press, 1982.

[36] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 5-229-256, May 1992.

Vijaykumar Gullapalli received the B.S. degree in electrical engineering and the M.S. degree in mathematics from the Birla Institute of Technology and Science, India, in 1984. He received the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, in 1988 and 1992, respectively. He currently holds a joint appointment with the UMass Adaptive Networks and Perceptual Robotics Laboratories as a Post-Doctoral Research Associate, pursuing robotic applications of his dissertational work on reinforcement learning and its application to control. His interests include learning in natural and artificial systems, algorithms for learning and adaptation in connectionist systems, and applications of connectionist learning techniques to problems in control, cognition, and pattern recognition. His current research focuses on learning control under conditions of uncertainty and noise.

Judy A. Franklin earned the B.A. degree in mathematics at Clarion University of Pennsylvania in 1980. She then studied at the University of Massachusetts, Amherst, where she earned the M.S. degree in computer science (1983) and the Ph.D. degree in electrical and computer engineering (1988). Her thesis focused on learning and compliance in robotics, using reinforcement learning and conventional control techniques. She is interested in dextrous machines as well as machines that are cognitively inclined (and both). Her current research is a melding of symbolic machine learning and artificial intelligence, artificial neural networks, and control techniques. The purpose is to control dynamic systems that are undertaking complex problems and tasks. She has been an originator and builder in two different hardware dynamic systems labs, one at UMass and one at GTE Laboratories Incorporated where she has worked since 1987. At GTE she is part of an Adaptive Systems Department that applies many different methodologies to GTE business problems, such as wireless communication channel allocation and network traffic control.

Hamid Benbrahim received the Maitrise in electrical engineering and D.E.A. in instrumentation and control from Université de Caen, France, in 1988 and 1989, respectively. He is currently doing research at GTE Laboratories Incorporated in machine learning and is attending the University of New Hampshire for a Ph.D. in electrical engineering. His dissertation will focus on real-time learning control applied to biped robots. His main interests lie in robotics and intelligent control.