Proceeding of the 2004 American Control Conference
Boston, Massachusetts June 30 - July 2, 2004

Reinforcement Learning With Supervision by a Stable Controller

Michael T. Rosenstein and Andrew G. Barto
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
mtr@cs.umass.edu, barto@cs.umass.edu

Abstract— **Reinforcement learning (RL) methods provide a means for solving optimal control problems when accurate models are unavailable. For many such problems, however, RL alone is impractical and the associated learning problem must be structured somehow to take advantage of prior knowledge. In this paper we examine the use of such knowledge in the form of a stable controller that generates control inputs in parallel with an RL system. The controller acts as a supervisor that not only teaches the RL system about favorable control actions but also protects the learning system from risky behavior. We demonstrate the approach with a simulated robotic arm and a real seven-DOF manipulator.**

I. INTRODUCTION

Reinforcement learning (RL) is an artificial intelligence paradigm for optimal control. With origins in animal learning research—especially trial-and-error learning—modern RL algorithms are built upon well established mathematical models for decision making and control. The primary advantage of RL over more traditional optimal control techniques is that accurate models, while useful for RL, are not required for optimal performance. But like other forms of intelligent control, RL has several drawbacks that limit its effectiveness when put into practice with realistic control problems. In particular, the time required for learning can be prohibitive, and system behavior during learning can lead to unacceptable risks. For this reason, in most large-scale applications RL is used with simulated rather than real dynamic systems. To overcome these difficulties, we propose a control framework that combines RL with conventional methods. More specifically, we use a stable, yet sub-optimal controller that serves as a kind of supervisor for the RL component of the framework.

Almost all RL methods that incorporate supervisory information, e.g., [1]–[7], do so by modifying a *value function*, which is typically used to store a learned ranking of the control actions available in a given state. The corresponding control policy is then represented implicitly, usually as the action with the best ranking for each state. The alternative described in this paper involves an *actor-critic architecture* for RL [8]. Actor-critic architectures differ from other value-based methods in that separate data structures are used for the control policy (the "actor") and the value function (the "critic"). The responsibility of the critic is to learn how the actor's behavior affects the performance objective, and this information, in turn, is useful for the actor to learn how to adjust its control policy toward optimality.

One important advantage of the actor-critic framework is that the explicitly represented policy can be modified directly by standard supervised learning methods. For instance, backpropagation can be used to update the actor when implemented as a multilayer artificial neural network. In any case, the actor can change its behavior based on training data provided by a supervisor, such as an easily designed controller, without the need to calculate the associated values of those data. The critic (or some other comparable mechanism) is still required for optimization, whereas the supervisor helps the actor achieve a level of proficiency whenever the critic has a poor estimate of the value function. In the next section we describe a *supervised* actor-critic architecture where the supervisor supplies not only error information for the actor to learn from, but also control actions for the plant.

II. SUPERVISED ACTOR-CRITIC ARCHITECTURE

Figure 1 shows a schematic of the usual actor-critic architecture [9] augmented by three major pathways for incorporating supervisor information. Along the "shaping" pathway, the supervisor supplies the critic a source of evaluative feedback, or *reward*, in addition to the rewards normally provided by the plant, or *environment*. Shaping essentially simplifies the task of the learning system by making good control actions easier to discriminate from bad ones. For instance, the critic may receive favorable evaluations for behavior which is only approximately correct given the

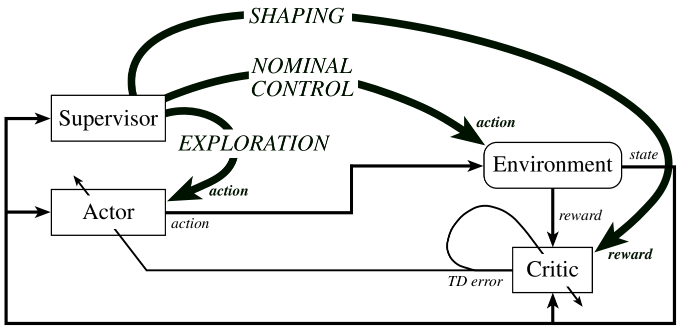

Fig. 1. Actor-critic architecture and several pathways for supervisor information.

original task. As the actor gains proficiency, the supervisor then gradually withdraws the additional reward feedback to shape the learned policy toward optimality for the true task. With "nominal control" the supervisor sends control signals, or *actions*, directly to the environment. For example, the supervisor may override bad commands from the actor as a way to ensure safety and to guarantee a minimum standard of performance. And along the "exploration" pathway, the supervisor provides the actor with hints about which actions may or may not be promising for the current situation, thereby altering the exploratory nature of the actor's trial-and-error learning. In this section, we focus on the latter two pathways, and we examine the use of supervised learning which offers a powerful counterpart to RL methods.

The combination of supervised learning with actor-critic RL was first suggested by Clouse and Utgoff [10] and independently by Benbrahim and Franklin [11]. Figure 2 shows our version of the supervised actor-critic architecture, which differs from previous work in a key way described in Section II-B. Taken together, the actor, the supervisor and the gain scheduler form a "composite" actor that sends a composite action to the environment. Note that we use "gain scheduling" in a broad sense to mean the blending of two or more sources of control actions, cf. [12]. The environment responds to the composite action with a transition from the current state, s, to the next state, s'. The environment also provides an evaluation called the immediate reward, r. The job of the critic is to observe states and rewards and to build a value function, $V^\pi(s)$, that accounts for both immediate and future rewards received under the composite policy, π. This value function is defined recursively by the Bellman equation,

$$V^\pi(s) = \sum_{s' \in \mathcal{S}} \Pr(s'|s,a)\{R(s') + \gamma V^\pi(s')\},$$

where $\mathcal{S}$ is the set of admissible states, $R(s')$ is the expected value of r, $\gamma \in [0,1]$ is a factor that discounts the value of the next state, and $\Pr(s'|s,a)$ is the probability of transitioning to state s' after executing action $a = \pi(s)$.

For RL problems, the expected rewards and the state-transition probabilities are typically unknown. Learning, therefore, must proceed from samples, i.e., from observed rewards and state transitions, and temporal-difference (TD)

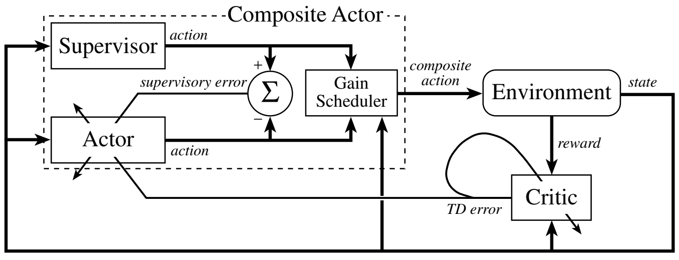

Fig. 2. The supervised actor-critic architecture.

methods [13] are commonly used to update the state-value *estimates*, $V(s)$, by an amount proportional to the TD error, which is defined as

$$\delta = r + \gamma V(s') - V(s).$$

Whenever the TD error is positive, the state of affairs is better than expected and so one increases the estimated value of state s. Similarly, when $\delta < 0$, the action chosen according to π resulted in a situation worse than expected and so one decreases $V(s)$. In short, TD methods improve past estimates of the value function by using future—typically more accurate—estimates of V.

A. The Gain Scheduler

For the examples in Section III, the gain scheduler computes the composite action, a, as simply a weighted sum of the actions given by the component policies. In particular,

$$a \leftarrow ka^E + (1-k)a^S,$$

where a^E is the actor's exploratory action and a^S is the supervisor's action, as given by policies π^E and π^S, respectively. (The supervisor's actions are observable but its policy is generally unknown.) We also denote by a^A the actor's greedy action determined by the corresponding policy, π^A. Typically, π^E is a copy of π^A modified to include an additive random variable with zero mean. Thus, each exploratory action is simply a noisy copy of the corresponding greedy action, although we allow for the possibility of more sophisticated exploration strategies.

The parameter $k \in [0,1]$ interpolates between π^E and π^S, and therefore k determines the level of control, or autonomy, on the part of the actor. In general, the value of k varies with state, although we drop the explicit dependence on s to simplify notation. The parameter k also plays an important role in modifying the actor's policy, as described in more detail below. We assume that π^A is given by a function approximator with the parameter vector w, and after each state transition, those parameters are updated according to a rule of the form

$$w \leftarrow w + k\Delta w^{\mathrm{RL}} + (1-k)\Delta w^{\mathrm{SL}}, \tag{1}$$

where Δw^{RL} and Δw^{SL} are the individual updates based on RL and supervised learning, respectively. Thus, k also interpolates between two styles of learning.

B. The Actor Update Equation

To make the reinforcement-based adjustment to the parameters of π^A we compute

$$\Delta w^{\mathrm{RL}} \leftarrow \alpha\delta(a^E - a^A)\nabla_w \pi^A(s), \tag{2}$$

where α is a step-size parameter. When the TD error is positive, this update will push the greedy policy evaluated at s closer to a^E, i.e., closer to the exploratory action which led to a state with estimated value better than expected.

Similarly, when $\delta < 0$, the update will push $\pi^A(s)$ away from a^E and in subsequent visits to state s the corresponding exploratory policy will select this unfavorable action with reduced probability.

To compute the supervised learning update, Δw^{SL}, we seek to minimize in each observed state the supervisory error

$$E = \frac{1}{2}[\pi^S(s) - \pi^A(s)]^2.$$

Locally, this is accomplished by following a steepest descent heuristic, i.e., by making an adjustment proportional to the negative gradient of the error with respect to w:

$$\Delta w^{\text{SL}} \leftarrow -\alpha \nabla_w E(s).$$

Expanding the previous equation with the chain rule and substituting the observed actions gives the usual kind of gradient descent learning rule:

$$\Delta w^{\text{SL}} \leftarrow \alpha(a^S - a^A)\nabla_w \pi^A(s). \tag{3}$$

Finally, by substituting Eqs. (2) and (3) into Eq. (1) we obtain the desired actor update equation:

$$w \leftarrow w + \alpha[k\delta(a^E - a^A) + (1-k)(a^S - a^A)]\nabla_w \pi^A(s). \tag{4}$$

Eq. (4) summarizes a steepest descent algorithm where k trades off between two sources of gradient information: one from a performance surface based on the evaluation signal and one from a quadratic error surface based on the supervisory error. A complete algorithm was presented in [14].

As mentioned above, the architecture shown in Figure 2 is similar to one suggested previously by Clouse and Utgoff [10] and by Benbrahim and Franklin [11]. However, our approach is novel in the following way: In the figure, we show a direct connection from the supervisor to the actor, whereas the supervisor in both [10] and [11] influences the actor indirectly through its effects on the environment as well as the TD error. Using our notation the corresponding update equation for these other approaches, e.g., [11, Eq. (1)], essentially becomes

$$w \leftarrow w + \alpha[k\delta(a^E - a^A) + (1-k)\delta(a^S - a^A)]\nabla_w \pi^A(s) \tag{5}$$

$$= w + \alpha\delta[ka^E + (1-k)a^S - a^A]\nabla_w \pi^A(s). \tag{6}$$

The key attribute of Eq. (5) is that the TD error modulates the supervisory error, $a^S - a^A$. This may be a desirable feature if one "trusts" the critic more than the supervisor, in which case one should view the supervisor as an additional source of exploration. However, Eq. (5) may cause the steepest descent algorithm to ascend the associated error surface, especially early in the learning process when the critic has a poor estimate of the true value function. Moreover, when δ is small, the actor loses the ability to learn from its supervisor, whereas in Eq. (4) this ability depends primarily on the interpolation parameter, k.

III. EXAMPLES

Previously, we demonstrated the benefits of our approach with a ship steering task, i.e., with a standard problem from the optimal control literature [14]. The examples in this paper demonstrate that the style of control and learning used for the ship steering task is also suitable for learning to exploit the dynamics of a robotic arm. Each of these examples is a targeting task where the supervisor is a stable controller that brings the system to goal, although in a sub-optimal fashion. Thus, the supervisor enables the composite actor in Figure 2 to solve the task *on the very first trial*, and on every trial while it improves, whereas the task is virtually impossible to solve with RL alone.

The implementation of the learning algorithm was given in [14], although several relevant details are repeated here. In particular, we implemented both actor and critic by a tile coding scheme, i.e., CMAC neural network [15], with a total of 25 tilings, or layers, per CMAC. To make the interaction between supervisor and actor dependent on state, the interpolation parameter, k, was set according to a state-visitation histogram, also implemented as a CMAC. During each step of the learning process, the value of k was set to the CMAC output for the current state (initially 0) with values cut off at a maximum of $k = 1$, i.e., at full autonomy. At the end of each trial, the weights from the "visited" histogram tiles were incremented by a small amount. Thus, the gain scheduler made a gradual shift from full supervision to full autonomy as the actor and critic acquired enough control knowledge to reach the goal reliably. A decay factor of 0.999 was also used to downgrade the weight of each CMAC tile; in effect, autonomy "leaked away" from infrequently visited regions of state space.

A. Simulated Robotic Arm

Our first example involves a simulated robotic arm that was modeled as a two-link pendulum with each link having length 0.5 m and mass 2.5 kg. The equations of motion [16] were integrated numerically using Euler's method with a step size of 0.001 s. Actions from both actor and supervisor were generated every 0.75 s and were represented as two-dimensional velocity vectors with joint speed limits of ± 0.5 rad/sec. The task was to move with minimum effort from the initial configuration with joint angles of -90 and 0 degrees to the goal configuration with joint angles of 135 and 90 degrees. For this demonstration, effort was quantified as the total integrated torque magnitude.

The supervisor in this example is a hand-crafted controller that moves the arm at maximum speed directly toward the goal in configuration space. Therefore, actions from the supervisor always lie on a unit square centered at the origin, whereas the actor is free to choose from the entire set of admissible actions. In effect, the supervisor's policy is to follow a straight-line path to the goal—which is time-optimal given the velocity constraints. Due to the dynamics of the robot, however, straight-line paths are not necessarily optimal with respect to other performance objectives, such as minimum energy.

A lower-level control system was responsible for transforming commanded velocities into motor torques for each joint. This occurred with a control interval of 0.001 s and in several stages: First, the commanded velocity was adjusted to account for acceleration constraints that eliminate abrupt changes in velocity, especially at the beginning and end of movement. The adjusted velocity, along with the current position, was then used to compute the desired position at the end of the next control interval. Third, a proportional-derivative (PD) controller converted this target position into joint torques, but with a target velocity of zero rather than the commanded velocity. And finally, a simplified model of the arm was used to adjust the feedback-based torque to include a feedforward term that compensates for gravity. This scheme is intended to match the way some industrial manipulators are controlled once given a higher-level movement command, e.g., velocity as used here. Gravity compensation guarantees stability of the lower-level controller [16], and the PD target velocity of zero helps ensure that the arm will stop safely given a communications failure with the higher level.

The above control scheme also holds an advantage for learning. Essentially, the manipulator behaves in accordance with a tracking controller—only the desired trajectory is revealed gradually with each control decision from the higher level. At this level, the manipulator behaves like an overdamped, approximately first-order system, and so policies need not account for the full state of the robot. That is, for both actor and supervisor it suffices to use reduced policies that map from positions to velocity commands, rather than policies that map from positions *and* velocities to acceleration commands. As is common with tracking controllers, this abstraction appears to cancel the dynamics we intend to exploit. However, by designing an optimal control problem, we allow the dynamics to influence the learning system by way of the performance objective, i.e., through the reward function.

For the RL version of this optimal control problem, the objective was to maximize the total discounted reward, with immediate rewards computed as the negative effort accumulated over each 0.75 s decision interval. The discount parameter in this example was set to $\gamma = 1$, i.e., no discounting. Exploratory actions, a^E, were Gaussian distributed with a mean equal to the greedy action, except that a^E was clipped at the joint speed limits. The standard deviation of the exploratory actions was initially 1.0 rad/sec, but this value decayed exponentially toward zero by a factor of 0.999 after each trial. CMAC tiles were uniform with a width of 25 degrees along each input dimension; the actor CMAC was initialized to zero whereas the critic CMAC was initialized to -300 (a pessimistic estimate of total negative effort).

Figure 3(a) shows the configuration of the robot every

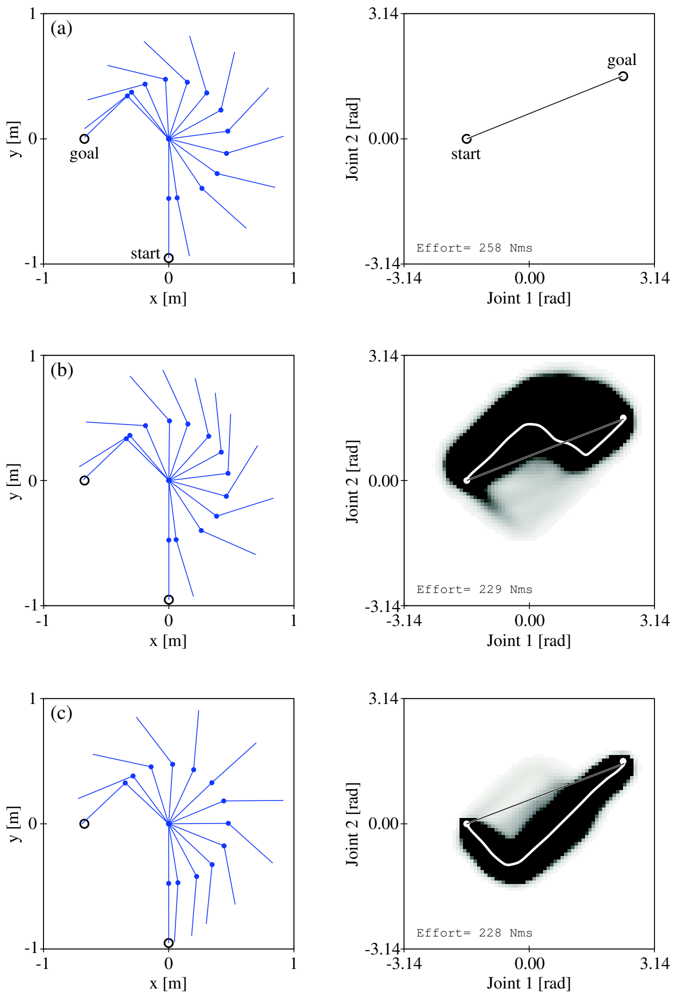

Fig. 3. Simulated two-link arm after (a) no learning and (b,c) 5000 learning trials. Configuration-space paths after learning are shown in white, and the grayscale region indicates the level of autonomy from the state-visitation histogram, with values ranging from $k = 0$ (white) to $k = 1$ (black).

0.75 s along a straight-line path to the goal. The proximal joint has more distance to cover and therefore moves at maximum speed, while the distal joint moves at a proportionately slower speed. The total effort for this fully supervised policy is 258 Nm·s. Figures 3(b) and 3(c) show examples of improved performance after 5000 trials of learning, with a final cost of 229 and 228 Nm·s, respectively. In each of the left-hand diagrams, the corresponding "spokes" from the proximal link fall in roughly the same position, and so the observed improvements are due to the way the distal joint modulates its movement around the straight-line path, as shown in the right-hand diagrams.

Figure 4 shows the effects of learning averaged over 25 runs. The value of the optimal policy for this task is unknown, although the best observed solution has a cost of 216 Nm·s. Most improvement happens within 400 trials and the remainder of learning shows a drop in variability

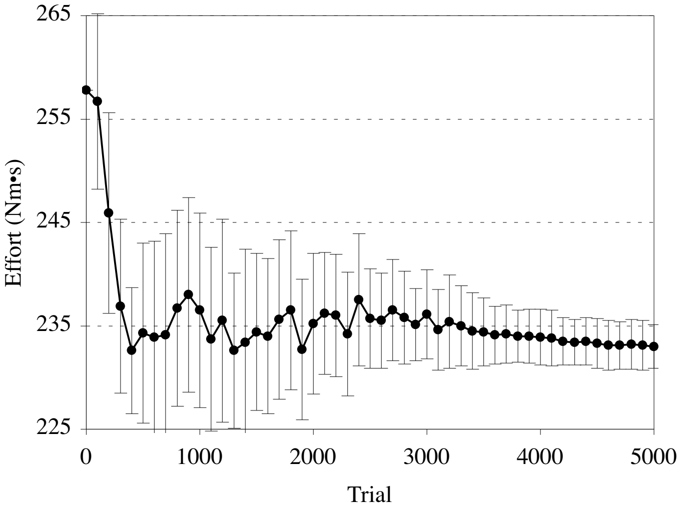

Fig. 4. Effects of learning for the simulated two-link arm averaged over 25 runs of 5000 trials each.

as the exploration policy "decays" toward the greedy control policy. One difficulty with this example is the existence of many locally optimal solutions to the task. This causes the learning system to wander among satisfactory solutions, with convergence to one of them only when forced to do so by the reduced exploration.

B. Case Study With a Real Robot

To demonstrate that the methods in this paper are suitable for real robots, we replicated the previous example with a seven degree-of-freedom whole arm manipulator (WAM; Barrett Technology Inc., Cambridge, MA). Figure 5 shows a sequence of several postures as the WAM moves from the start configuration (far left frame) to the goal configuration (far right frame). As with the previous example the task was formulated as a minimum-effort optimal control problem—utilizing a stable tracking controller and a supervisor that generates straight-line trajectories to the goal in configuration space. The joint speed limits for this example were increased to ± 0.75 rad/sec rather than ± 0.5 rad/sec as used above. The learning algorithm was virtually identical to the one in the previous example, although several parameter values were modified to encourage reasonable improvement with very few learning trials. For instance, the histogram increment was increased slightly, thereby facilitating a faster transition to autonomous behavior. Also, the level of exploration did not decay, but rather remained constant, and a^E was Gaussian distributed with a standard deviation of 0.25 rad/sec.

Figure 6 shows the effects of learning averaged over 5 runs. Performance worsens during the first 10 to 20 trials due to the initialization of the actor's policy. More specifically, at the start of learning the actor's policy maps all inputs to the zero velocity vector, and so the actor cannot move the robot until it has learned how to do so from its supervisor. The drawback of this initialization scheme—along with a fast transition to autonomous behavior—is that early in the learning process the supervisor's commands become diminished when blended with the actor's near-zero commands. The effect is slower movement of the manipulator and prolonged effort while raising the arm against gravity. However, after 60 trials of learning the supervised actor-critic architecture shows statistically significant improvement ($p < 0.01$) over the supervisor alone. After 120 trials, the overall effect of learning is approximately 20% reduced effort despite an increased average movement time from 4.16 s to 4.34 s (statistically significant with $p < 0.05$).

IV. CONCLUSIONS

The examples in Section III demonstrate a gradual shift from full supervision to full autonomy—blending two sources of actions and learning feedback. Much like the examples by Clouse [1] and by Maclin and Shavlik [4], this shift happens in a state-dependent way with the actor seeking help from the supervisor in unfamiliar territory. Unlike these other approaches, the actor also clones the supervisor's control policy very quickly over the visited states. This style of learning is similar to methods that seed an RL system with training data, e.g., [7], [17], although with the supervised actor-critic architecture, the interpolation parameter allows the seeding to happen in an incremental fashion at the same

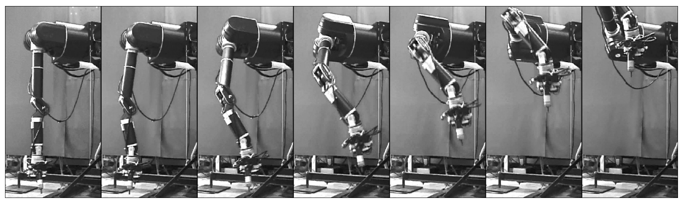

Fig. 5. Representative configurations of the WAM after learning.

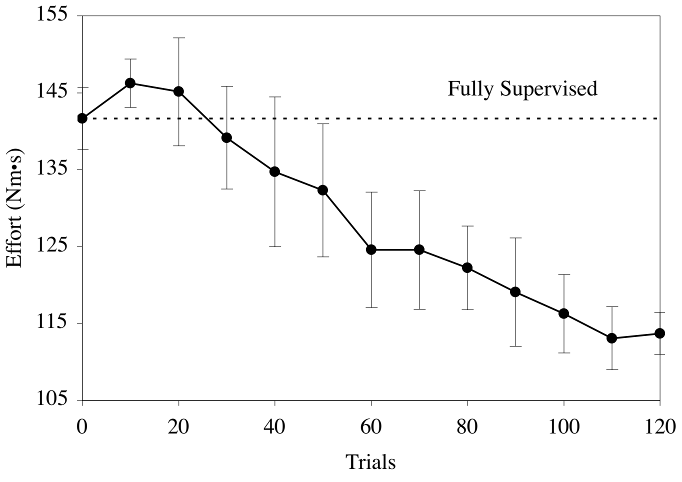

Fig. 6. Effects of learning for the WAM averaged over 5 runs of 120 trials each.

time as trial-and-error learning.

One drawback of these methods for control of real robots is the time needed for training. By most standards in the RL literature, the supervised actor-critic architecture requires relatively few trials, at least for the examples presented above. However, some robot control problems may permit extremely few learning trials, say 10 or 20. Clearly, in such cases we should not expect optimality; instead we should strive for methods that provide gains commensurate with the training time. In any case, we might tolerate slow optimization if we can deploy a learning robot with provable guarantees on the worst-case performance. Recent work by Kretchmar *et al.* [5] and by Perkins and Barto [6] demonstrates initial progress in this regard.

Despite the challenges when we combine RL with supervised learning, we still reap benefits from both paradigms. From actor-critic architectures we gain the ability to discover behavior that optimizes performance. From supervised learning we gain a flexible way to incorporate prior knowledge. In particular, the internal representations used by the actor can be very different from those used by the supervisor. The actor, for example, can be an artificial neural network, while the supervisor can be expert knowledge encoded as logical propositions, a conventional feedback controller as demonstrated in this paper, or even a human supplying actions that depend on an entirely different perception of the environment's state. Presumably the supervisor has a certain proficiency at a given task, which the actor exploits for improved performance throughout learning.

ACKNOWLEDGMENTS

We thank Andrew Fagg, Mohammad Ghavamzadeh, and Balaraman Ravindran for helpful discussions. This work was supported by NASA under award No. NAG-9-1379. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Aeronautics and Space Administration.

REFERENCES

[1] J. A. Clouse, "On integrating apprentice learning and reinforcement learning," Ph.D. dissertation, University of Massachusetts, Amherst, 1996.

[2] M. Dorigo and M. Colombetti, "Robot shaping: developing autonomous agents through learning," *Artificial Intelligence*, vol. 71, no. 2, pp. 321–370, 1994.

[3] M. Huber and R. A. Grupen, "A feedback control structure for on-line learning tasks," *Robotics and Autonomous Systems*, vol. 22, no. 3–4, pp. 303–315, 1997.

[4] R. Maclin and J. W. Shavlik, "Creating advice-taking reinforcement learners," *Machine Learning*, vol. 22, no. (1-3), pp. 251–281, 1996.

[5] R. M. Kretchmar, P. M. Young, C. W. Anderson, D. C. Hittle, M. L. Anderson, C. C. Delnero, and J. Tu, "Robust reinforcement learning control with static and dynamic stability," *International Journal of Robust and Nonlinear Control*, vol. 11, pp. 1469–1500, 2001.

[6] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 803–832, 2002.

[7] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*. Piscataway, NJ: IEEE, 2002, pp. 3404–3410.

[8] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, pp. 835–846, 1983.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press, 1998.

[10] J. A. Clouse and P. E. Utgoff, "A teaching method for reinforcement learning," in *Proceedings of the Nineth International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, 1992, pp. 92–101.

[11] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, pp. 283–302, 1997.

[12] J. S. Shamma, "Linearization and gain-scheduling," in *The Control Handbook*, W. S. Levine, Ed. Boca Raton, FL: CRC Press, 1996, pp. 388–396.

[13] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[14] M. T. Rosenstein and A. G. Barto, "Supervised learning combined with an actor-critic architecture," Department of Computer Science, University of Massachusetts, Tech. Rep. 02–41, 2002.

[15] J. S. Albus, *Brains, Behavior, and Robotics*. Peterborough, NH: Byte Books, 1981.

[16] J. J. Craig, *Introduction To Robotics : Mechanics and Control*. Reading, MA: Addison-Wesley Publishing Company, 1989.

[17] S. Schaal, "Learning from demonstration," in *Advances In Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. Cambridge, MA: The MIT Press, 1997, pp. 1040–1046.