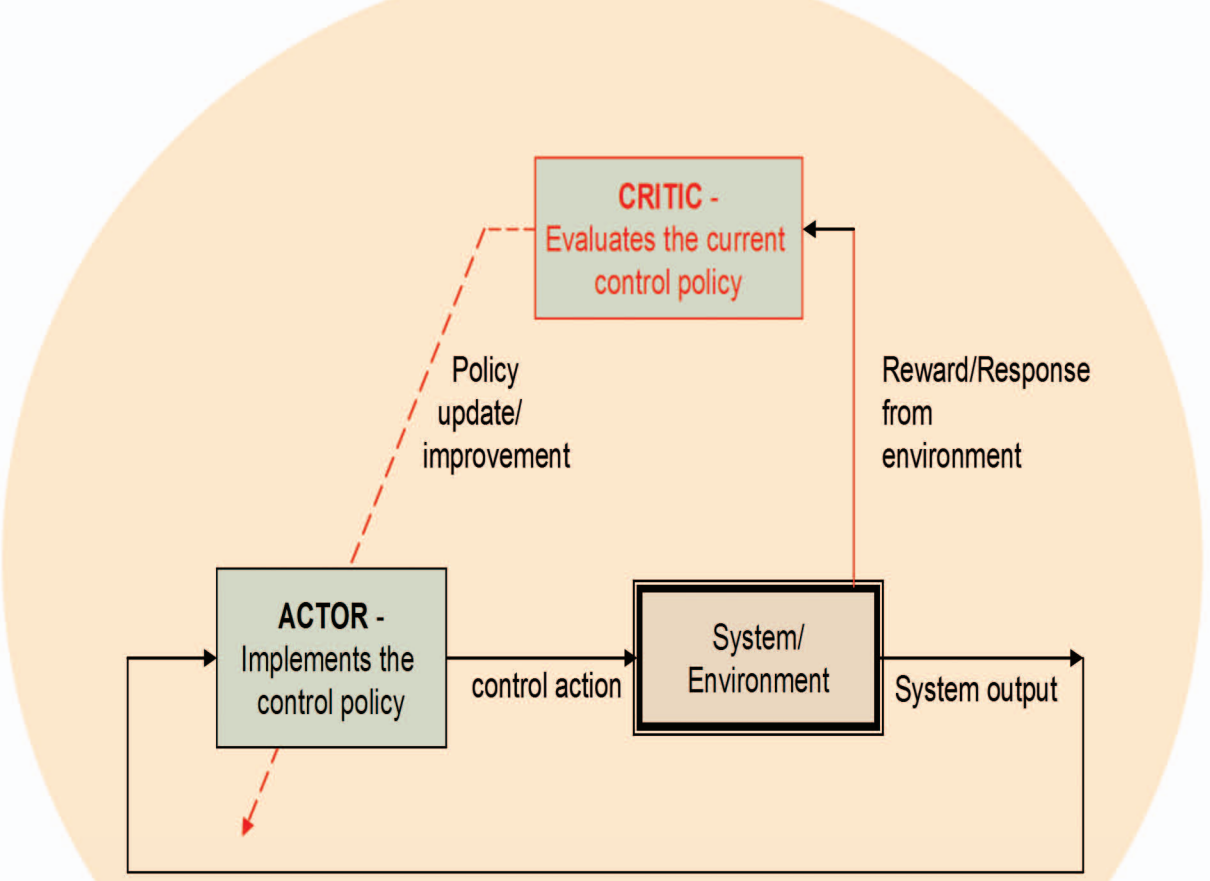

Reinforcement Learning and Feedback Control

USING NATURAL DECISION METHODS TO DESIGN OPTIMAL ADAPTIVE CONTROLLERS

FRANK L. LEWIS, DRAGUNA VRABIE,
and KYRIAKOS G. VAMVOUDAKIS

This article describes the use of principles of reinforcement learning to design feedback controllers for discrete- and continuous-time dynamical systems that combine features of adaptive control and optimal control. Adaptive control [1], [2] and optimal control [3] represent different philosophies for designing feedback controllers. Optimal controllers are normally designed offline by solving Hamilton–Jacobi–Bellman (HJB) equations, for example, the Riccati equation, using complete knowledge of the system dynamics. Determining optimal control policies for nonlinear systems

Digital Object Identifier 10.1109/MCS.2012.2214134
Date of publication: 12 November 2012

1066-033X/12/$31.00©2012IEEE

requires the offline solution of nonlinear HJB equations, which are often difficult or impossible to solve. By contrast, adaptive controllers learn online to control unknown systems using data measured in real time along the system trajectories. Adaptive controllers are not usually designed to be optimal in the sense of minimizing user-prescribed performance functions. Indirect adaptive controllers use system identification techniques to first identify the system parameters and then use the obtained model to solve optimal design equations [1]. Adaptive controllers may satisfy certain inverse optimality conditions [4].

This article shows that the technique known as *reinforcement learning* allows for the design of a class of adaptive controllers with actor-critic structure that learn optimal control solutions by solving HJB design equations online, forward in time, and without knowing the full system dynamics. In the linear quadratic case, these methods determine the solution to the algebraic Riccati equation online, without specifically solving the Riccati equation and without knowing the system state matrix A. As such, these controllers can be considered as being optimal adaptive controllers. Chapter 11 of [3] places these controllers in the context of optimal control systems.

Reinforcement learning is a type of machine learning developed in the computational intelligence community in computer science and engineering. It has close connections to both optimal control and adaptive control. More specifically, reinforcement learning refers to a class of methods that enable the design of adaptive controllers that learn online, in real time, the solutions to user-prescribed optimal control problems. Reinforcement learning methods were used by Ivan Pavlov in the 1860s to train his dogs. In machine learning, reinforcement learning [5]–[9] is a method for solving optimization problems that involves an actor or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions. Reinforcement learning is inspired by natural learning mechanisms, where animals adjust their actions based on reward and punishment stimuli received from the environment [10], [11]. Other reinforcement learning mechanisms operate in the human brain, where the dopamine neurotransmitter in the basal ganglia acts as a reinforcement informational signal that favors learning at the level of the neuron [12]–[15].

Reinforcement learning implies a cause-and-effect relationship between actions and reward or punishment. It implies goal-directed behavior, at least insofar as the agent has an understanding of reward versus lack of reward or punishment. The reinforcement learning algorithms are constructed on the idea that effective control decisions must be remembered, by means of a reinforcement signal, such that they become more likely to be used a second time. Reinforcement learning is based on real-time evaluative information from the environment and could be called *action-based learning*. Reinforcement learning is connected from a theoretical point of view with both adaptive control and optimal control methods.

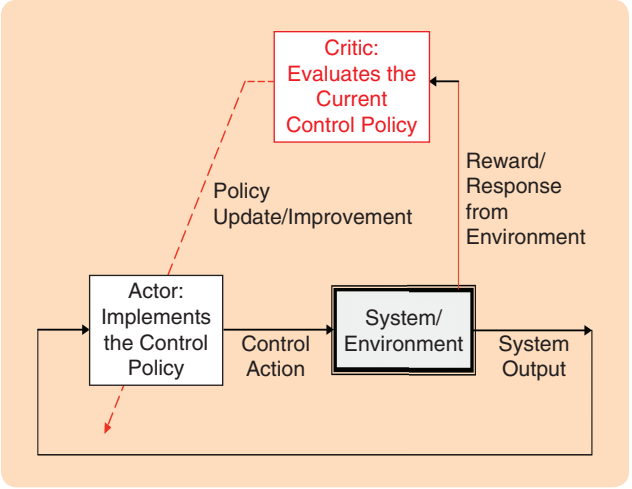

FIGURE 1 Reinforcement learning with an actor/critic structure. This structure provides methods for learning optimal control solutions online based on data measured along the system trajectories.

One type of reinforcement learning algorithms employs the actor-critic structure shown in Figure 1 [16]. This structure produces forward-in-time algorithms that are implemented in real time wherein an actor component applies an action, or control policy, to the environment, and a critic component assesses the value of that action. The learning mechanism supported by the actor-critic structure has two steps, namely, policy evaluation, executed by the critic, followed by policy improvement, performed by the actor. The policy evaluation step is performed by observing from the environment the results of applying current actions. These results are evaluated using a performance index, or value function [5], [6], [11], [17], [18], that quantifies how close the current action is to optimal. Performance or value can be defined in terms of optimality objectives such as minimum fuel, minimum energy, minimum risk, or maximum reward. Based on the assessment of the performance, one of several schemes can then be used to modify or improve the control policy in the sense that the new policy yields a value that is improved relative to the previous value. In this scheme, reinforcement learning is a means of learning optimal behaviors by observing the real-time responses from the environment to nonoptimal control policies.

Werbos [7], [14], [19] developed actor-critic techniques for feedback control of discrete-time dynamical systems that learn optimal policies online in real time using data measured along the system trajectories. These methods, known as approximate dynamic programming (ADP) or adaptive dynamic programming, comprise a family of

This article presents the main ideas and algorithms of reinforcement learning and their applications to feedback control of dynamical systems.

four basic learning methods. The ADP controllers are actor-critic structures with one learning network for the control action and one learning network for the critic. Many surveys of ADP are available [20]–[24]. Bertsekas and Tsitsiklis developed reinforcement learning methods for the control of discrete-time dynamical systems [17]. This approach, known as *neurodynamic programming*, used offline solution methods. ADP has been extensively used in feedback control applications. Applications have been reported for missile control [25], automotive control [26], aircraft control over a flight envelope [27], aircraft landing control [20], [28], [29], helicopter reconfiguration after rotor failure [30], power system control [31], and vehicle steering and speed control [32]. Convergence analyses of ADP are available [31], [33], [34].

One framework for studying reinforcement learning is based on Markov decision processes (MDPs). Many dynamical decision problems can be formulated as MDPs including feedback control systems for human-engineered systems, feedback regulation mechanisms for population balance and survival of species [35], [36], decision-making in multiplayer games, and economic mechanisms for regulation of global financial markets.

This article presents the main ideas and algorithms of reinforcement learning and their applications to feedback control of dynamical systems. We start from a discussion of MDP and develop the Bellman equation, upon which rest many reinforcement learning methods. Policy iteration and value iteration [5], [6], [11] are presented, and it is described how they relate to dynamic programming [18], which is a backward-in-time method for computing optimal controllers. We focus next on temporal difference methods to show how reinforcement learning leads to a family of optimal adaptive controllers for discrete-time systems. These adaptive controllers have an actor-critic structure and as such learn the solutions to optimal control problems online in real time. Applications of reinforcement learning for feedback control of continuous-time systems have been impeded by the inconvenient form of the continuous-time Hamiltonian, which contains the system dynamics. Descriptions are given on how to use a method known as integral reinforcement learning [15], [37] to circumvent this problem and design a class of optimal adaptive controllers for continuous-time systems. These methods enable the solution of HJB design equations to be solved online and forward in time without knowing the full system dynamics.

The optimal adaptive controllers presented in this article are a natural extension of adaptive controllers. Direct adaptive controllers tune the controller parameters. Indirect adaptive controllers identify a model for the system, and the identified model is then used in design equations to compute a controller. Optimal adaptive controllers based on the actor-critic structure are a logical extension of this sequence in that they identify the performance value of the current control policy, and then use that information to update the controller.

Note that in computational intelligence, the control action is applied by an agent to the system, which is interpreted to be the environment. By contrast, in control system engineering, the control action is interpreted as being applied to a system or plant that represents the vehicle, process, or device being controlled. This difference captures the differences in philosophy between reinforcement learning in computational intelligence and in feedback control systems design.

MARKOV DECISION PROCESSES

MDPs provide a framework for studying reinforcement learning. This section reviews MDP [5], [11], [17], starting by defining optimal sequential decision problems, where decisions are made at stages of a process evolving through time. Dynamic programming is next presented, which provides methods for solving optimal decision problems by working backward through time. Dynamic programming is an offline solution technique that cannot be implemented online in a forward-in-time fashion. Reinforcement learning and adaptive control are concerned with determining control solutions in real time and forward in time. The key to this approach is provided by the Bellman equation, which is then described. The subsequent section describes methods known as *policy iteration* and *value iteration* that provide algorithms based on the Bellman equation for solving optimal decision problems in real time forward in time based on data measured along the system trajectories.

Consider the MDP (X, U, P, R), where X is a set of states and U is a set of actions or controls. The transition probabilities $P: X \times U \times X \to [0, 1]$ describe, for each state $x \in X$ and action $u \in U$, the conditional probability $P^u_{x,x'} = \Pr\{x' \mid x, u\}$ of transitioning to state $x' \in X$ given the MDP is in state x and takes action u. The cost function $R: X \times U \times X \to R$ is the expected immediate cost $R^u_{xx'}$ paid after transition to state $x' \in X$ given that the MDP starts in state $x \in X$ and takes action $u \in U$. The Markov

property refers to the fact that transition probabilities $P^u_{x,x'}$ depend only on the current state x and not on the history of how the MDP attained that state.

The basic problem for MDP is to find a mapping $\pi: X \times U \to [0, 1]$ that gives, for each state x and action u, the conditional probability $\pi(x, u) = \Pr\{u \mid x\}$ of taking action u given that the MDP is in state x. Such a mapping is referred to as a closed-loop control or action *strategy* or *policy*. The strategy or policy $\pi(x, u) = \Pr\{u \mid x\}$ is called *stochastic* or *mixed* if there is a nonzero probability of selecting more than one control when in state x. Mixed strategies can be viewed as probability distribution vectors having as component i the probability of selecting the ith control action while in state $x \in X$. If the mapping $\pi: X \times U \to [0, 1]$ admits only one control, with probability one, when in every state x, the mapping is called a *deterministic* policy. Then, $\pi(x, u) = \Pr\{u \mid x\}$ corresponds to a function mapping states into controls $\mu(x): X \to U$.

MDPs that have finite state and action spaces are termed *finite MDPs*.

Optimal Sequential Decision Problems

Dynamical systems evolve causally through time. We consider sequential decision problems and impose a discrete stage index k such that the MDP takes an action and changes states at nonnegative integer stage values k. The stages may correspond to time or more generally to sequences of events. We refer to the stage value as the time. Denote state values and actions at time k by x_k, u_k . MDPs evolve in discrete time.

It is often desirable for human-engineered systems to be optimal in terms of conserving resources such as cost, time, fuel, and energy. Thus, the notion of optimality should be captured in selecting control policies for MDPs. Define a *stage cost* at time k by $r_k = r_k(x_k, u_k, x_{k+1})$. Then $R^u_{xx'} = E\{r_k \mid x_k = x, u_k = u, x_{k+1} = x'\}$, with $E\{\cdot\}$ the expected value operator. Define a performance index as the sum of future costs over the time interval $[k, k + T]$,

$$J_{k,T} = \sum_{i=0}^{T} \gamma^i r_{k+i} = \sum_{i=k}^{k+T} \gamma^{i-k} r_i, \tag{1}$$

where $0 \le \gamma < 1$ is a discount factor that reduces the weight of costs incurred further in the future.

The usage of MDPs in the fields of computational intelligence and economics usually considers r_k as a reward incurred at time k, also known as *utility*, and $J_{k,T}$ as a discounted return, also known as a *strategic reward*. This article instead refers to *stage costs* and *discounted future costs* to be consistent with objectives in the control of dynamical systems.

Consider that an agent selects a control policy $\pi_k(x_k, u_k)$ that is used at each stage k of the MDP. We are primarily interested in *stationary policies*, where the conditional probabilities $\pi_k(x_k, u_k)$ are independent of k. Then $\pi_k(x, u) = \pi(x, u) = \Pr\{u \mid x\}$, for all k. Nonstationary deterministic policies have the form $\pi = \{\mu_0, \mu_1, \cdots\}$, where each entry is a function $\mu_k(x): X \to U; k = 0, 1, \ldots$. Stationary deterministic policies are independent of time, that is, have the form $\pi = \{\mu, \mu, \cdots\}$.

Select a fixed stationary policy $\pi(x, u) = \Pr\{u \mid x\}$. Then the "closed-loop" MDP reduces to a Markov chain with state space X. That is, the transition probabilities between states are fixed with no further freedom of choice of actions. The transition probabilities of this Markov chain are given by

$$p_{x,x'} \equiv P^{\pi}_{x,x'} = \sum_u \Pr\{x' \mid x, u\} \Pr\{u \mid x\} = \sum_u \pi(x, u) P^u_{x,x'}, \tag{2}$$

where the Chapman-Kolmogorov identity [38] is used.

A Markov chain is *ergodic* if all states are positive recurrent and aperiodic [38]. Under the assumption that the Markov chain corresponding to each policy, with transition probabilities given in (2), is ergodic, it can be shown that every MDP has a stationary deterministic optimal policy [17], [39]. Then, for a given policy, there exists a stationary distribution $p_\pi(x)$ over X that gives the steady-state probability the Markov chain is in state x.

The *value* of a policy is defined as the conditional expected value of future cost when starting in state x at time k and following policy $\pi(x, u)$ thereafter,

$$V^{\pi}_k(x) = E_\pi\{J_{k,T} \mid x_k = x\} = E_\pi\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\right\}, \tag{3}$$

where $E_\pi\{\cdot\}$ is the expected value given that the agent follows policy $\pi(x, u)$, and $V^\pi(x)$ is known as the *value function* for policy $\pi(x, u)$, which is the value of being in state x given that the policy is $\pi(x, u)$.

A main objective of MDP is to determine a policy $\pi(x, u)$ to minimize the expected future cost

$$\begin{aligned} \pi^*(x, u) &= \arg\min_\pi V^{\pi}_k(s) \\ &= \arg\min_\pi E_\pi\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\right\}. \end{aligned} \tag{4}$$

This policy is termed the *optimal policy*, and the corresponding *optimal value* is given as

$$V^*_k(x) = \min_\pi V^{\pi}_k(x) = \min_\pi E_\pi\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\right\}. \tag{5}$$

In computational intelligence and economics, the interest is in utilities and rewards, and the interest is in *maximizing* the expected performance index.

A Backward Recursion for the Value

By using the Chapman-Kolmogorov identity and the Markov property, the value of the policy $\pi(x,u)$ can be written as

$$V_k^\pi(x) = E_\pi\{J_k \mid x_k = x\} = E_\pi\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\right\}, \quad (6)$$

$$V_k^\pi(x) = E_\pi\left\{r_k + \gamma \sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i \mid x_k = x\right\}, \quad (7)$$

$$V_k^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma E_\pi\left\{\sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i \mid x_{k+1} = x'\right\}\right]. \quad (8)$$

Therefore the value function for the policy $\pi(x,u)$ satisfies

$$V_k^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^\pi(x')]. \quad (9)$$

This equation provides a backward recursion for the value at time k in terms of the value at time $k+1$.

Dynamic Programming

The optimal cost can be written as

$$V_k^*(x) = \min_\pi V_k^\pi(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^\pi(x')]. \quad (10)$$

Bellman's optimality principle [18] states that "An optimal policy has the property that no matter what the previous control actions have been, the remaining controls constitute an optimal policy with regard to the state resulting from those previous controls." This principle implies that (10) can be written as

$$V_k^*(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^*(x')]. \quad (11)$$

Suppose an arbitrary control u is now applied at time k, and the optimal policy is applied from time $k+1$ on. Then Bellman's optimality principle indicates that the optimal control policy at time k is given by

$$\pi^*(x,u) = \arg\min_\pi \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^*(x')]. \quad (12)$$

Under the assumption that the Markov chain corresponding to each policy, with transition probabilities given in (2), is ergodic, every MDP has a stationary deterministic optimal policy. Then we can equivalently minimize the conditional expectation over all actions u in state x. Therefore,

$$V_k^*(x) = \min_u \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^*(x')], \quad (13)$$

$$u_k^* = \arg\min_u \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^*(x')]. \quad (14)$$

The backward recursion (11), (13) forms the basis for dynamic programming [18], which gives offline methods for working *backward in time* to determine optimal policies [3]. DP is an offline procedure for finding the optimal value and optimal policies that requires knowledge of the complete system dynamics in the form of transition probabilities $P_{x,x'}^u = \Pr\{x' \mid x, u\}$ and expected costs $R_{xx'}^u = E\{r_k \mid x_k = x, u_k = u, x_{k+1} = x'\}$.

Bellman Equation and Bellman Optimality Equation

Dynamic programming is a backward-in-time method for finding the optimal value and policy. By contrast, reinforcement learning is concerned with finding optimal policies based on causal experience by executing sequential decisions that improve control actions based on the observed results of using a current policy. This procedure requires the derivation of methods for finding optimal values and optimal policies that can be executed forward in time. The key to this is the Bellman equation. References for this section include [5]–[7], [11], and [16].

To derive forward-in-time methods for finding optimal values and optimal policies, set the time horizon T to infinity and define the infinite-horizon cost

$$J_k = \sum_{i=0}^{\infty} \gamma^i r_{k+i} = \sum_{i=k}^{\infty} \gamma^{i-k} r_i. \quad (15)$$

The associated infinite-horizon value function for the policy $\pi(x,u)$ is

$$V^\pi(x) = E_\pi\{J_k \mid x_k = x\} = E_\pi\left\{\sum_{i=k}^{\infty} \gamma^{i-k} r_i \mid x_k = x\right\}. \quad (16)$$

By using (8) with $T = \infty$, it is seen that the value function for the policy $\pi(x,u)$ satisfies the *Bellman equation*

$$V^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V^\pi(x')]. \quad (17)$$

The key to deriving this equation is that the same value function appears on both sides, which is due to the fact that the infinite-horizon cost is used. Therefore, the Bellman equation (17) can be interpreted as a consistency equation that must be satisfied by the value function at each time stage. It expresses a relation between the current value of being in state x and the value of being in next state x' given that policy $\pi(x,u)$ is used. The solution to the Bellman equation is the value given by the infinite sum in (16).

The Bellman equation (17) is the starting point for developing a family of reinforcement learning algorithms for finding optimal policies by using causal experiences received stagewise forward in time. The Bellman optimality equation (11) involves the "minimum" operator and so does not contain any specific policy $\pi(x,u)$. Its solution relies on knowing the dynamics, in the form of transition probabilities. By contrast, the form of the Bellman equation is simpler than that of the optimality equation, and it is easier to solve. The solution to the Bellman equation yields the value function of a specific policy $\pi(x,u)$. As such, the Bellman equation is well suited to the actor-critic method of reinforcement learning shown in Figure 1. It is shown

subsequently that the Bellman equation provides methods for implementing the critic in Figure 1, which is responsible for evaluating the performance of the specific current policy. Two key ingredients remain to be put in place. First, it is shown that methods known as policy iteration and value iteration use the Bellman equation to solve optimal control problems forward in time. Second, by approximating the value function in (17) by a parametric structure, these methods can be implemented online using standard adaptive control system identification algorithms such as recursive least-squares.

In the context of using the Bellman equation (17) for reinforcement learning, $V^\pi(x)$ may be considered as a predicted performance, $\sum_u \pi(x,u)\sum_{x'} P^u_{xx'} R^u_{xx'}$ the observed one-step reward, and $V^\pi(x')$ as a current estimate of future behavior. Such notions will be used in the subsequent discussion of temporal difference learning to develop adaptive control algorithms that can learn optimal behavior online in real-time applications.

If the MDP is finite and has N states, then the Bellman equation (17) is a system of N simultaneous linear equations for the value $V^\pi(x)$ of being in each state x given the current policy $\pi(x,u)$.

The optimal value satisfies

$$V^*(x) = \min_\pi V^\pi(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^\pi(x')]. \quad (18)$$

Bellman's optimality principle then yields the *Bellman optimality equation*

$$V^*(x) = \min_\pi V^\pi(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^*(x')]. \quad (19)$$

Equivalently, under the ergodicity assumption on the Markov chains corresponding to each policy, the Bellman optimality equation can be written as

$$V^*(x) = \min_u \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^*(x')]. \quad (20)$$

This equation is known as the HJB equation in control systems. If the MDP is finite and has N states, then the Bellman optimality equation is a system of N nonlinear equations for the optimal value $V^*(x)$ of being in each state. The optimal control is given by

$$u^* = \arg\min_u \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^*(x')]. \quad (21)$$

These equations can be written in the context of feedback control of dynamical systems. "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" shows that, for the linear quadratic regulator (LQR), the Bellman equation (17) becomes a Lyapunov equation. "The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation" shows that the Bellman optimality equation (19) becomes an algebraic Riccati equation in the LQR case.

POLICY EVALUATION AND POLICY IMPROVEMENT

Given a current policy $\pi(x,u)$, its value (16) can be determined by solving the Bellman equation (17). This procedure is known as *policy evaluation*. Moreover, given the value for some policy $\pi(x,u)$, we can always use it to find another policy that is better, or at least no worse. This step is known as *policy improvement*. Specifically, suppose $V^\pi(x)$ satisfies (17). Then define a new policy $\pi'(x,u)$ by

$$\pi'(x,u) = \arg\min_\pi \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^\pi(x')]. \quad (22)$$

Then it can be shown that $V^{\pi'}(x) \le V^\pi(x)$ [5], [17]. The policy determined as in (22) is said to be *greedy* with respect to value function $V^\pi(x)$.

In the special case that $V^{\pi'}(x) = V^\pi(x)$ in (22), then $V^{\pi'}(x)$, $\pi'(x,u)$ satisfy (20), (21). Therefore $\pi'(x,u) = \pi(x,u)$ is the optimal policy and $V^{\pi'}(x) = V^\pi(x)$ the optimal value. That is, an optimal policy, and only an optimal policy, is greedy with respect to its own value. In computational intelligence, *greedy* refers to quantities determined by optimizing over short or one-step horizons, without regard to potential impacts far into the future.

Now consider algorithms that repeatedly interleave the two procedures:

Policy Evaluation by Bellman Equation

$$V^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^\pi(x')], \quad \text{for all } x \in S \subseteq X. \quad (23)$$

Policy Improvement

$$\pi'(x,u) = \arg\min_\pi \sum_{x'} P^u_{xx'}[R^u_{xx'} + \gamma V^\pi(x')], \quad \text{for all } x \in S \subseteq X \quad (24)$$

where S is a suitably selected subspace of the state space, to be discussed in more detail later. An application of (23) followed by an application of (24) is referred to as one *step*. This terminology is in contrast to the decision time stage k defined above.

At each step of such algorithms, a policy is obtained that is no worse than the previous policy. Therefore, it is not difficult to prove convergence under fairly mild conditions to the optimal value and optimal policy. Most such proofs are based on the Banach fixed point theorem. Note that (20) is a fixed point equation for $V^*(\cdot)$. Then the two equations (23), (24) define an associated map that can be shown under mild conditions to be a contraction map [6], [17], [40] that converges to the solution of (20).

Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation

MDP DYNAMICS FOR DETERMINISTIC DISCRETE-TIME SYSTEMS

Consider the discrete-time linear quadratic regulator (LQR) problem, where the MDP is deterministic and satisfies the state transition equation

$$x_{k+1} = Ax_k + Bu_k, \tag{S1}$$

with the discrete time index k. The associated infinite-horizon performance index has deterministic stage costs and is

$$J_k = \frac{1}{2}\sum_{i=k}^{\infty} r_i = \tfrac{1}{2}\sum_{i=k}^{\infty}(x_i^T Qx_i + u_i^T Ru_i). \tag{S2}$$

In this example, the state space $X = R^n$ and action space $U = R^m$ are infinite and continuous.

THE BELLMAN EQUATION FOR DISCRETE-TIME LQR IS A LYAPUNOV EQUATION

Select a policy $u_k = \mu(x_k)$ and write the associated value function as

$$V(x_k) = \frac{1}{2}\sum_{i=k}^{\infty} r_i = \frac{1}{2}\sum_{i=k}^{\infty}(x_i^T Qx_i + u_i^T Ru_i). \tag{S3}$$

An equivalent difference equation is

$$\begin{aligned} V(x_i) &= \frac{1}{2}(x_k^T Qx_k + u_k^T Ru_k) + \frac{1}{2}\sum_{i=k+1}^{\infty}(x_i^T Qx_i + u_i^T Ru_i) \\ &= \frac{1}{2}(x_k^T Qx_k + u_k^T Ru_k) + V(x_{k+1}). \end{aligned} \tag{S4}$$

That is, the solution $V(x_k)$ to this equation that satisfies $V(0) = 0$ is the value given by (S3). Equation (S4) is exactly the Bellman equation (17) for the LQR.

Assuming that the value is quadratic in the state so that

$$V_k(x_k) = \frac{1}{2}x_k^T Px_k, \tag{S5}$$

for some kernel matrix P, yields the Bellman equation form

$$2V(x_k) = x_k^T Px_k = x_k^T Qx_k + u_k^T Ru_k + x_{k+1}^T Px_{k+1}, \tag{S6}$$

which, using the state equation, can be written

$$2V(x_k) = x_k^T Qx_k + u_k^T Ru_k + (Ax_k + Bu_k)^T P(Ax_k + Bu_k). \tag{S7}$$

Assuming a constant, that is, stationary, state feedback policy $u_k = \mu(x_k) = -Kx_k$ for some stabilizing gain K, write

$$\begin{aligned} 2V(x_k) &= x_k^T Px_k \\ &= x_k^T Qx_k + x_k^T K^T RKx_k \\ &\quad + x_k^T (A - BK)^T P(A - BK)x_k. \end{aligned} \tag{S8}$$

Since this equation holds for all state trajectories, we have

$$(A - BK)^T P(A - BK) - P + Q + K^T RK = 0, \tag{S9}$$

which is a Lyapunov equation. That is, the Bellman equation (17) for the discrete-time LQR is equivalent to a Lyapunov equation. Since the performance index is undiscounted, that is, $\gamma = 1$, a stabilizing gain K, that is, a stabilizing policy, must be selected.

The formulations (S4), (S6), (S8), and (S9) for the Bellman equation are all equivalent. Note that forms (S4) and (S6) do not involve the system dynamics (A, B). On the other hand, note that the Lyapunov equation (S9) can only be used if the state dynamics (A, B) are known. Optimal control design using the Lyapunov equation is the standard procedure in control systems theory. Unfortunately, by assuming that (S8) holds for all trajectories and going to (S9), we lose all possibility of applying any sort of reinforcement learning algorithms to solve for the optimal control and value online by observing data along the system trajectories. By contrast, we show that by employing the form (S4) or (S6) for the Bellman equation, reinforcement learning algorithms for learning optimal solutions online can be devised by using temporal difference methods. That is, reinforcement learning allows the Lyapunov equation to be solved online without knowing A or B.

A large family of algorithms is available that implements the policy evaluation and policy improvement procedures in different ways, or interleaves them differently, or select the subspace $S \subseteq X$ in different ways, to determine the optimal value and optimal policy. Some of these algorithms are outlined later in this article.

The relevance of this discussion for feedback control systems is that these two procedures can be implemented for dynamical systems online in real time by observing data measured along the system trajectories. The result is a family of adaptive control algorithms that converge to optimal control solutions. Such algorithms are of the *actor-critic* class of reinforcement learning systems, shown in Figure 1. There, a critic agent evaluates the current control policy using methods based on (23). After this evaluation is completed, the action is updated by an actor agent based on (24).

Policy Iteration

One method of reinforcement learning for using (23), (24) to find the optimal value and optimal policy is *policy iteration*.

Policy Iteration Algorithm

Select an initial policy $\pi_0(x, u)$. Starting with $j = 0$, iterate on j until convergence:

The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation

The discrete-time LQR Hamiltonian function is

$$H(x_k,u_k) = x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T \times P(Ax_k + Bu_k) - x_k^T P x_k. \quad \text{(S10)}$$

The Hamiltonian is equivalent to the temporal difference error in MDP. A necessary condition for optimality is the stationarity condition $\partial H(x_k,u_k)/\partial u_k = 0$, which is equivalent to (22). Solving this equation yields the optimal control

$$u_k = -Kx_k = -(B^T P B + R)^{-1} B^T P A x_k.$$

Inserting this equation into (S8) yields the discrete-time algebraic Riccati equation (ARE)

$$A^T P A - P + Q - A^T P B (B^T P B + R)^{-1} B^T P A = 0. \quad \text{(S11)}$$

The ARE is exactly the Bellman optimality equation (19) for the discrete-time LQR.

Policy Evaluation (Value Update)

$$V_j(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} [R^u_{xx'} + \gamma V_j(x')], \quad \text{for all } x \in X. \quad (25)$$

Policy Improvement (Policy Update)

$$\pi_{j+1}(x,u) = \arg\min_\pi \sum_{x'} P^u_{xx'} [R^u_{xx'} + \gamma V_j(x')], \quad \text{for all } x \in X. \quad (26)$$

At each step j, the policy iteration algorithm determines the solution of the Bellman equation (25) to compute the value $V_j(x)$ of using the current policy $\pi_j(x,u)$. This value corresponds to the infinite sum (16) for the current policy. Then the policy is improved using (26). The steps are continued until there is no change in the value or the policy.

Note that j is not the time or stage index k but a policy iteration step iteration index. As detailed in the next sections, policy iteration can be implemented for dynamical systems online in real time by observing data measured along the system trajectories. Data for multiple times k are needed to solve the Bellman equation (25) at each step j.

The policy iteration algorithm must be suitably initialized to converge. The initial policy $\pi_0(x,u)$ and value V_0 must be selected so that $V_1 \leq V_0$. Then, for finite Markov chains with N states, policy iteration converges in a finite number of steps, less than or equal to N, because there are only a finite number of policies [17].

If the MDP is finite and has N states, then the policy evaluation equation (25) is a system of N simultaneous linear equations, one for each state. Instead of directly solving the Bellman equation (25), it can be solved by an iterative policy evaluation procedure. Note that (25) is a fixed point equation for $V_j(\cdot)$ that defines the *iterative policy evaluation* map

$$V_j^{i+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} [R^u_{xx'} + \gamma V_j^i(x')], \quad i = 1, 2, \ldots, \quad (27)$$

which can be shown to be a contraction map under rather mild conditions. By the Banach fixed point theorem, the iteration can be initialized at any nonnegative value of $V_j^1(\cdot)$ and the iteration converges to the solution of (25). Under certain conditions, this solution is unique. A suitable initial value choice is the value function $V_{j-1}(\cdot)$ from the previous step $j-1$. On close enough convergence, set $V_j(\cdot) = V_j^i(\cdot)$ and proceed to apply (26).

The index j in (27) refers to the step number of the policy iteration algorithm. By contrast, i is an iteration index. Iterative policy evaluation (27) should be compared to the backward-in-time recursion (9) for the finite-horizon value. In (9), k is the time index. By contrast, in (27), i is an iteration index. Dynamic programming is based on (9) and proceeds backward in time. The methods for online optimal adaptive control described in this article proceed forward in time and are based on policy iteration and similar algorithms.

Value Iteration

A second method for using (23), (24) in reinforcement learning is *value iteration*.

Value Iteration Algorithm

Select an initial policy $\pi_0(x,u)$. Starting with $j = 0$, iterate on j until convergence:

Value Update

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} [R^u_{xx'} + \gamma V_j(x')], \quad \text{for all } x \in S_j \subseteq X. \quad (28)$$

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_\pi \sum_{x'} P^u_{xx'} [R^u_{xx'} + \gamma V_{j+1}(x')], \quad \text{for all } x \in S_j \subseteq X. \quad (29)$$

The value update and policy improvement can be combined into one equation to obtain the equivalent form for value iteration

$$V_{j+1}(x) = \min_{\pi} \sum_{u} \pi(x,u) \sum_{x'} P^{u}_{xx'} [R^{u}_{xx'} + \gamma V_j(x')], \quad \text{for all } x \in S_j \subseteq X. \tag{30}$$

or, equivalently under the ergodicity assumption, in terms of deterministic policies

$$V_{j+1}(x) = \min_{u} \sum_{x'} P^{u}_{xx'} [R^{u}_{xx'} + \gamma V_j(x')], \quad \text{for all } x \in S_j \subseteq X. \tag{31}$$

Note that now (28) is a simple one-step recursion, not a system of linear equations as is (25) in the policy iteration algorithm. In fact, value iteration uses one iteration of (27) in its value update step. It does not find the value corresponding to the current policy but takes only one iteration toward that value. Again, j is not the time index, but the value iteration step index.

Subsequent sections describe how to implement value iteration for dynamical systems online in real time by observing data measured along the system trajectories. Data for multiple times k are needed to solve the update (28) for each step j.

Standard value iteration takes the update set as $S_j = X$, for all j. That is, the value and policy are updated for all states simultaneously. *Asynchronous* value iteration methods perform the updates on only a subset of the states at each step. In the extreme case, updates can be performed on only one state at each step.

It is shown in [17] that standard value iteration, which has $S_j = X$, for all j, converges for a finite MDP for all initial conditions when the discount factor satisfies $0 < \gamma < 1$. When $S_j = X$, for all j and $\gamma = 1$, an absorbing state is added and a "properness" assumption is needed to guarantee convergence to the optimal value. When a single state is selected for value and policy updates at each step, the algorithm converges, for all choices of initial value, to the optimal cost and policy if each state is selected for update infinitely often. More universal algorithms result if the value update (28) is performed multiple times for different choices of S_j prior to a policy improvement. Then, it is required that updates (28) and (29) be performed infinitely often for each state, and a monotonicity assumption must be satisfied by the initial starting value.

Considering (19) as a fixed point equation, value iteration is based on the associated iterative map (28), (29), which can be shown under certain conditions to be a contraction map. In contrast to policy iteration, which converges under certain conditions in a finite number of steps, value iteration usually takes an infinite number of steps to converge [17]. Consider finite MDP, and consider the transition probability graph having probabilities (2) for the Markov chain corresponding to an optimal policy $\pi^*(x,u)$. If this graph is acyclic for some $\pi^*(x,u)$, then value iteration converges in at most N steps when initialized with a large value.

Having in mind the dynamic programming equation (9) and examining the value iteration value update (28), $V_j(x')$ can be interpreted as an approximation or estimate for the future stage cost to go from the future state x'. Those algorithms wherein the future cost estimates are themselves costs or values for some policy are called *rollout algorithms* in [17]. Such policies are forward looking and self-correcting. These methods can be used to derive algorithms for receding horizon control [41].

MDP, policy iteration, and value iteration are closely tied to optimal and adaptive control. "Policy Iteration and Value Iteration for the Discrete-Time LQR" shows that for the discrete-time LQR, policy iteration and value iteration can be used to derive algorithms for solution of the optimal control problem that are quite common in the feedback control systems, including Hewer's algorithm.

Generalized Policy Iteration

In policy iteration the system of linear equations (25) is completely solved at each step to compute the value (16) of using the current policy $\pi_j(x,u)$. This solution can be accomplished by running iterations (27) until convergence. By contrast, in value iteration only one iteration of (27) is taken in the value update step (28). *Generalized* policy iteration algorithms make several iterations (27) in their value update step.

Usually, policy iteration converges to the optimal value in fewer steps j since it does more work in solving equations at each step. On the other hand, value iteration is the easiest to implement as it only takes one iteration of a recursion in (28). Generalized policy iteration provides a suitable compromise between computational complexity and convergence speed. Generalized policy iteration is a special case of the value iteration algorithm given above, where we select $S_j = X$, for all j and perform value update (28) multiple times before each policy update (29).

Q Function

The conditional expected value in (13),

$$Q^*_k(x,u) = \sum_{x'} P^{u}_{xx'} [R^{u}_{xx'} + \gamma V^*_{k+1}(x')] = E_{\pi}\{r_k + \gamma V^*_{k+1}(x') \mid x_k = x, u_k = u\}, \tag{32}$$

is known as the *optimal Q function* [42], [43]. The letter Q comes from "quality function." The Q function is also called

Policy Iteration and Value Iteration for the Discrete-Time LQR

The Bellman equation (17) for the discrete-time LQR is equivalent to all the formulations (S4), (S6), (S8), (S9) in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation." Any of these formulations can be used to implement policy iteration and value iteration.

POLICY ITERATION, HEWER'S ALGORITHM

With step index j, and using superscripts to denote algorithm steps and subscripts to denote the time k, the iterative policy evaluation step (25) applied on (S4) in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" yields

$$V^{j+1}(x_k) = \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + V^{j+1}(x_{k+1}). \tag{S12}$$

Policy iteration applied on (S6) yields

$$x_k^T P^{j+1} x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P^{j+1} x_{k+1}, \tag{S13}$$

and policy iteration on (S9) yields the Lyapunov equation

$$0 = (A - BK^j)^T P^{j+1}(A - BK^j) - P^{j+1} + Q + (K^j)^T R K^j. \tag{S14}$$

In all cases the policy improvement step is

$$\begin{aligned}\mu^{j+1}(x_k) &= K^{j+1} x_k \\ &= \arg\min (x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P^{j+1} x_{k+1}),\end{aligned} \tag{S15}$$

which can be written explicitly as

$$K^{j+1} = -(B^T P^{j+1} B + R)^{-1} B^T P^{j+1} A. \tag{S16}$$

The policy iteration algorithm format (S14), (S16) relies on repeated solutions of Lyapunov equations at each step and is Hewer's algorithm. This algorithm is proven to converge in [44] to the solution of the Riccati equation (S11) in "The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation." Hewer's algorithm is an offline algorithm that requires complete knowledge of the system dynamics (A, B) to find the optimal value and control. The algorithm requires that the initial gain K^0 be stabilizing.

VALUE ITERATION AND LYAPUNOV RECURSIONS

Applying value iteration (28) to Bellman equation format (S6) in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" yields

$$x_k^T P^{j+1} x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P_{k+1}^j, \tag{S17}$$

and on format (S9) in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" yields the Lyapunov recursion

$$P^{j+1} = (A - BK^j)^T P^j (A - BK^j) + Q + (K^j)^T R K^j. \tag{S18}$$

In both cases the policy improvement step is still given by (S15), (S16).

The value iteration algorithm format (S16), (S18) is a Lyapunov recursion, which is easy to implement and does not, in contrast to policy iteration, require Lyapunov equation solutions. This algorithm is shown to converge in [45] to the solution of the Riccati equation (S11) in "The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation." Lyapunov recursion is an offline algorithm that requires complete knowledge of the system dynamics (A, B) to find the optimal value and control. This algorithm does not require that the initial gain be stabilizing and can be initialized with any feedback gain.

ONLINE SOLUTION OF THE RICCATI EQUATION WITHOUT KNOWING THE PLANT MATRIX A

Hewer's algorithm and the Lyapunov recursion algorithm are both offline methods for solving the algebraic Riccati equation (S11) in "The Bellman Optimality Equation for Discrete-Time LQR is an Algebraic Riccati Equation." Full knowledge of the plant dynamics (A, B) is needed to implement these algorithms. By contrast, both the policy iteration algorithm format (S13), (S15) and the value iteration algorithm format (S17), (S15) can be implemented online to determine the optimal value and control in real time using data measured along the system trajectories,and without knowing the system matrix A. This aim is accomplished through the *temporal difference methods* described in the text. That is, reinforcement learning allows the solution of the algebraic Riccati equation online without knowing the system matrix A.

Iterative Policy Evaluation

Given a fixed policy K, the iterative policy evaluation procedure (27) becomes

$$P^{j+1} = (A - BK)^T P^j (A - BK) + Q + K^T R K. \tag{S19}$$

This recursion converges to the solution to the Lyapunov equation $P^{j+1} = (A - BK)^T P (A - BK) + Q + K^T R K$ if $(A - BK)$ is stable, for any choice of initial value P^0.

the *action-value function* [5]. The Q function is equal to the expected return for taking an arbitrary action u at time k in state x and thereafter following an optimal policy. The Q function is a function of the current state x and the action u.

In terms of the Q function, the Bellman optimality equation has the particularly simple form

$$V_k^*(x) = \min_u Q_k^*(x, u), \tag{33}$$

Q Function for the Discrete-Time LQR

The Q function following a given policy $u_k = \mu(x_k)$ is defined in (35). For the discrete-time LQR in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation," the Q function is

$$Q(x_k, u_k) = \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + V(x_{k+1}), \tag{S20}$$

where, the control u_k is arbitrary and the policy $u_k = \mu(x_k)$ is followed for $k+1$ and subsequent times. Writing

$$Q(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P (Ax_k + Bu_k), \tag{S21}$$

with P being the Riccati solution, yields the Q function for the discrete-time LQR:

$$Q(x_k, u_k) = \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} A^T PA + Q & B^T PA \\ A^T PB & B^T PB + R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}. \tag{S22}$$

Define

$$Q(x_k, u_k) \equiv \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T S \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} S_{xx} & S_{xu} \\ S_{ux} & S_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \tag{S23}$$

for kernel matrix S.

Applying $\partial Q(x_k, u_k)/\partial u_k = 0$ to (S23) yields

$$u_k = -S_{uu}^{-1} S_{ux} x_k, \tag{S24}$$

and to (S22) yields

$$u_k = -(B^T PB + R)^{-1} B^T PA x_k. \tag{S25}$$

The latter equation requires knowledge of the system dynamics (A, B) to perform the policy improvement step of either policy iteration or value iteration. On the other hand, (S24) requires knowledge only of the Q function matrix kernel S. "Adaptive Controller for Online Solution of Discrete-Time LQR Using Q Learning" shows how to use reinforcement learning temporal difference methods to determine the kernel matrix S online in real time without knowing the system dynamics (A, B) using data measured along the system trajectories. This procedure provides a family of Q learning algorithms that can solve the algebraic Riccati equation online without knowing the system dynamics (A, B).

$$u_k^* = \arg\min_u Q_k^*(x, u). \tag{34}$$

Given some fixed policy $\pi(x, u)$, define the Q function for that policy as

$$\begin{aligned} Q_k^\pi(x, u) &= E_\pi\{r_k + \gamma V_{k+1}^\pi(x') \mid x_k = x, u_k = u\} \\ &= \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V_{k+1}^\pi(x')], \end{aligned} \tag{35}$$

where (9) is used. This function is equal to the expected return for taking an arbitrary action u at time k in state x and thereafter following the existing policy $\pi(x, u)$. The meaning of the Q function is elucidated by "Q Function for the Discrete-Time LQR."

Since $V_k^\pi(x) = Q_k^\pi(x, \pi(x, u))$, (35) can be written as the backward recursion in the Q function:

$$Q_k^\pi(x, u) = \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma Q_{k+1}^\pi(x', \pi(x', u'))]. \tag{36}$$

The Q function is a function of both the current state x and the action u. By contrast, the value function is a function of the state. For finite MDP, the Q function can be stored as a lookup table for each state/action pair. Note that direct minimization in (11), (12) requires knowledge of the state transition probabilities, which correspond to the system dynamics, and costs. By contrast, the minimization in (33), (34) requires knowledge only of the Q function and not the system dynamics.

The utility of the Q function is twofold. First, it contains information about control actions in every state. As such, the best control in each state can be selected using (34) by knowing only the Q function. Second, the Q function can be estimated *online in real time* directly from date observed along the system trajectories, without knowing the system dynamics information, that is, the transition probabilities. The implementation of this online real-time estimation is described later in this article.

The infinite-horizon Q function for a prescribed fixed policy is given by

$$Q^\pi(x, u) = \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma V^\pi(x')]. \tag{37}$$

The Q function also satisfies a Bellman equation. Given a fixed policy $\pi(x, u)$,

$$V^\pi(x) = Q^\pi(x, \pi(x, u)), \tag{38}$$

hence according to (37) the Q function satisfies the Bellman equation

$$Q^\pi(x, u) = \sum_{x'} P_{xx'}^u [R_{xx'}^u + \gamma Q^\pi(x', \pi(x', u'))], \tag{39}$$

the Bellman optimality equation for the Q function is

$$Q^*(x,u) = \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma Q^*(x', \pi^*(x', u'))\right], \tag{40}$$

$$Q^*(x,u) = \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma \min_{u'} Q^*(x', u')\right]. \tag{41}$$

Compare (20) and (41), where the minimum operator and the expected value operator are reversed.

Policy iteration and value iteration are especially easy to implement in terms of the Q function (35), as follows.

Policy Iteration Using the Q Function

Policy Evaluation (Value Update)

$$Q_j(x,u) = \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma Q_j(x', \pi(x', u'))\right], \text{ for all } x \in X. \tag{42}$$

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_u Q_j(x,u), \text{ for all } x \in X. \tag{43}$$

Value Iteration Using the Q Function

Value Update

$$Q_{j+1}(x,u) = \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma Q_j(x', \pi(x', u'))\right], \quad \text{for all } x \in S_j \subseteq X. \tag{44}$$

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_u Q_{j+1}(x,u), \text{ for all } x \in S_j \subseteq X. \tag{45}$$

Combining both steps of value iteration yields the form

$$Q_{j+1}(x,u) = \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma \min_{u'} Q_j(x', u')\right], \quad \text{for all } x \in S_j \subseteq X, \tag{46}$$

which may be compared to (31).

As shown below, the utility of the Q function is that these algorithms can be implemented online in real time, without knowing the system dynamics, by measuring data along the system trajectories. These algorithms are an implementation of optimal adaptive control, that is, adaptive control algorithms that converge online to optimal control solutions.

Methods for Implementing Policy Iteration and Value Iteration

Multiple methods are available for performing the value and policy updates for policy iteration and value iteration [5], [6], [17]. The main three methods are exact computation, Monte Carlo methods, and temporal difference learning. The last two methods can be implemented without knowledge of the system dynamics. Temporal difference learning, which is covered in the next section, is the means by which optimal adaptive control algorithms can be derived for dynamical systems.

Policy iteration requires the solution at each step of Bellman equation (25) for the value update. For a finite MDP with N states, this is a set of linear equations in N unknowns, namely, the values of each state. Value iteration requires performing the one-step recursive update (28) at each step for the value update. Both of these iterations can be accomplished exactly if the transition probabilities $P^u_{xx'} = \Pr\{x' \mid x, u\}$ and costs $R^u_{xx'}$ of the MDP are known, which corresponds to knowing full system dynamics information. Likewise, the policy improvements (26), (29) can be explicitly computed if the dynamics are known. It is shown in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" and "The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation" that, for the discrete-time LQR, the exact computation method for computing the optimal control yields the Riccati equation solution approach. In this case policy iteration and value iteration are repetitive solutions of Lyapunov equations or Lyapunov recursions. In fact, policy iteration becomes Hewer's method [44], and the value iteration becomes the Lyapunov recursion scheme that is known to converge [45]. These techniques are offline methods that rely on matrix equation solutions and require complete knowledge of the system dynamics.

Monte Carlo learning is based on the definition (16) for the value function and uses repeated measurements of data to approximate the expected value. The expected values are approximated by averaging repeated results along sample paths. An assumption on the ergodicity of the Markov chain with transition probabilities (2) for the given policy being evaluated is implicit. This assumption is suitable for *episodic tasks*, with experience divided into episodes [5], namely, processes that start in an initial state and run until termination and are then restarted at a new initial state. For finite MDP, Monte Carlo methods converge to the true value function if all states are visited infinitely often. Therefore, to ensure accurate approximations of value functions, the episode sample paths must go through all the states $x \in X$ many times. This issue is called the problem of *maintaining exploration*. Several methods are available to ensure this amount of exploration, with one method being the use of *exploring starts*, in which every state has nonzero probability of being selected as the initial state of an episode.

Monte Carlo techniques are useful for dynamic control because the episode sample paths can be interpreted as system trajectories beginning in a prescribed initial

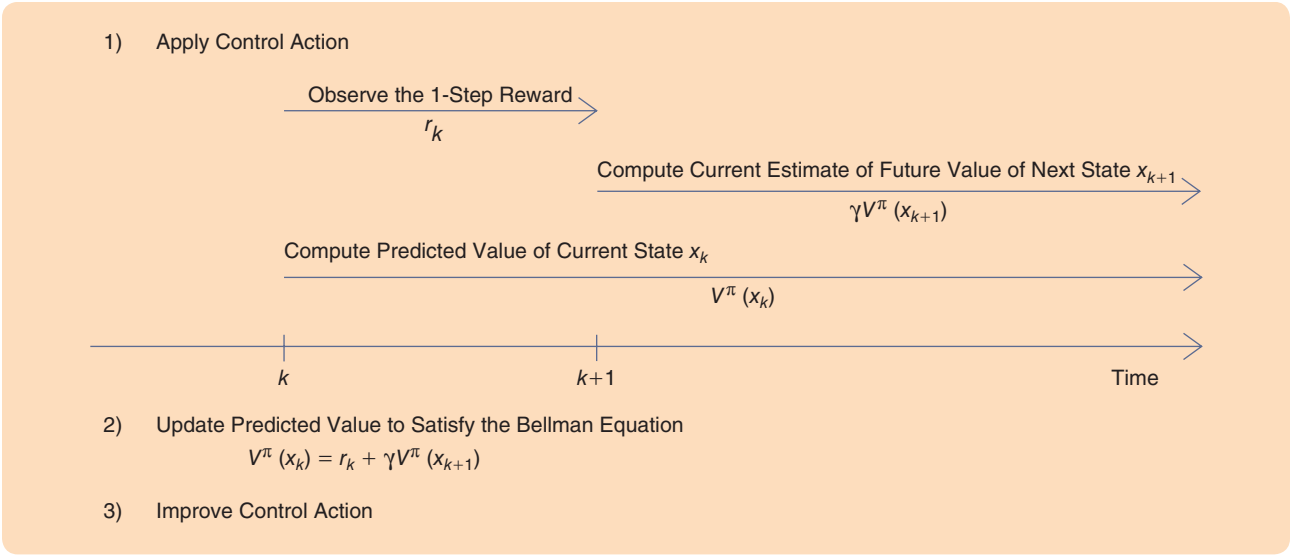

FIGURE 2 Temporal difference interpretation of the Bellman equation that shows how use of the Bellman equation captures the action, observation, evaluation, and improvement mechanisms of reinforcement learning.

state. However, no updates to the value function estimate or the control policy are made until after an episode terminates. In fact, Monte Carlo learning methods are closely related to repetitive or iterative learning control [46]. These methods do not learn in real time along a trajectory but learn as trajectories are repeated.

TEMPORAL DIFFERENCE LEARNING AND OPTIMAL ADAPTIVE CONTROL

It is now shown that the temporal difference method [5] for solving Bellman equations leads to a family of optimal adaptive controllers, that is, adaptive controllers that learn online the solutions to optimal control problems without knowing the full system dynamics. Temporal difference learning is true online reinforcement learning, wherein control actions are improved in real time based on estimating their value functions by observing data measured along the system trajectories.

Temporal Difference Learning Along State Trajectories

Policy iteration requires the solution at each step of N linear equations (25). Value iteration requires performing the recursion (28) at each step. Temporal difference reinforcement learning methods are based on the Bellman equation and solve equations such as (25), (28) without using systems dynamics knowledge, but using data observed along a single trajectory of the system. Therefore, temporal difference learning is applicable for feedback control applications. Temporal difference updates the value at each time step as observations of data are made along a trajectory. Periodically, the new value is used to update the policy. Temporal difference methods are related to adaptive control in that they adjust values and actions online in real time along system trajectories.

Temporal difference methods can be considered to be *stochastic approximation* techniques where by the Bellman equation (17), or its variants (25), (28), is replaced by its evaluation along a single sample path of the MDP. Then, the Bellman equation becomes a deterministic equation that allows the definition of a *temporal difference error.*

Equation (9) is used to write the Bellman equation (17) for the infinite-horizon value (16). According to (7)–(9), an alternative form for the Bellman equation is

$$V^{\pi}(x_k) = E_{\pi}\{r_k \mid x_k\} + \gamma E_{\pi}\{V^{\pi}(x_{k+1}) \mid x_k\}. \tag{47}$$

This equation forms the basis for temporal difference learning.

Temporal difference reinforcement learning uses one sample path, namely the current system trajectory, to update the value. Then, (47) is replaced by the *deterministic Bellman equation*

$$V^{\pi}(x_k) = r_k + \gamma V^{\pi}(x_{k+1}), \tag{48}$$

which holds for each observed data experience set (x_k, x_{k+1}, r_k) at each time stage k. This data set consists of the current state x_k, the observed cost incurred r_k, and the next state x_{k+1}. The *temporal difference error* is defined as

$$e_k = -V^{\pi}(x_k) + r_k + \gamma V^{\pi}(x_{k+1}), \tag{49}$$

and the value estimate is updated to make the temporal difference error small.

In the context of temporal difference learning, the interpretation of the Bellman equation is shown in Figure 2, where $V^{\pi}(x_k)$ may be considered as a predicted performance or value, r_k as the observed one-step reward, and $\gamma V^{\pi}(x_{k+1})$ as a current estimate of future

value. The Bellman equation can be interpreted as a consistency equation that holds if the current estimate for the predicted value $V^{\pi}(x_k)$ is correct. Temporal difference methods update the predicted value estimate $\hat{V}^{\pi}(x_k)$ to make the temporal difference error small. The idea, based on stochastic approximation, is that if we use the deterministic version of Bellman's equation repeatedly in policy iteration or value iteration, then on average these algorithms converge toward the solution of the stochastic Bellman equation.

OPTIMAL ADAPTIVE CONTROL FOR DISCRETE-TIME SYSTEMS

A family of optimal adaptive control algorithms can now be described for dynamical systems. These algorithms determine the solutions to HJ design equations online in real time without knowing the system drift dynamics. In the LQR case, this means that the algorithms solve the Riccati equation online without knowing the system matrix A. Physical analysis of dynamical systems using Lagrangian mechanics or Hamiltonian mechanics produces system descriptions in terms of nonlinear ordinary differential equations. Discretization yields nonlinear difference equations. Most research in reinforcement learning is conducted for systems that operate in discrete time [5], [14], [21], [39], so discrete-time dynamical systems are covered first, followed by continuous-time systems.

Temporal difference learning is a stochastic approximation technique based on the deterministic Bellman equation (48). Therefore, little is lost by considering deterministic systems here. Consider a class of discrete-time systems described by deterministic nonlinear dynamics in the affine state space difference equation form

$$x_{k+1} = f(x_k) + g(x_k)u_k, \tag{50}$$

with state $x_k \in R^n$ and control input $u_k \in R^m$. This form is used because its analysis is convenient. The following development can be generalized to the sampled-data form $x_{k+1} = F(x_k, u_k)$.

A deterministic control policy is defined as a function from state space to control space $h(\cdot): R^n \to R^m$. That is, for every state x_k, the policy defines a control action

$$u_k = h(x_k). \tag{51}$$

That is, a policy is a feedback controller.

Define a deterministic cost function that yields the value function

$$V^h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i) = \sum_{i=k}^{\infty} \gamma^{i-k} (Q(x_i) + u_i^T R u_i), \tag{52}$$

with $0 < \gamma \leq 1$ a discount factor, $Q(x_k) > 0$, $R > 0$, and $u_k = h(x_k)$ is a prescribed feedback control policy. That is, the stage cost is

$$r(x_k, u_k) = Q(x_k) + u_k^T R u_k. \tag{53}$$

The stage cost is taken as quadratic in u_k to simplify the development but can be any positive-definite function of the control. Assume that the system is stabilizable on some set $\Omega \in R^n$, that is, there exists a control policy $u_k = h(x_k)$ such that the closed-loop system $x_{k+1} = f(x_k) + g(x_k)h(x_k)$ is asymptotically stable on Ω. A control policy $u_k = h(x_k)$ is said to be *admissible* if it is stabilizing and yields a finite cost $V^h(x_k)$.

For the deterministic value (52), the optimal value is given by the Bellman optimality equation,

$$V^*(x_k) = \min_{h(\cdot)} (r(x_k, h(x_k)) + \gamma V^*(x_{k+1})), \tag{54}$$

which is just the discrete-time HJB equation. The optimal policy is then given as

$$h^*(x_k) = \arg\min_{h(\cdot)} (r(x_k, h(x_k)) + \gamma V^*(x_{k+1})). \tag{55}$$

In this setup, the deterministic Bellman's equation (48) is

$$\begin{aligned} V^h(x_k) &= r(x_k, u_k) + \gamma V^h(x_{k+1}) \\ &= Q(x_k) + u_k^T R u_k + \gamma V^h(x_{k+1}), V^h(0) = 0, \end{aligned} \tag{56}$$

which is a difference equation equivalent to the value in (52). That is, instead of evaluating the infinite sum (52), the difference equation (56) can be solved, with boundary condition $V(0) = 0$, to obtain the value of using a current policy $u_k = h(x_k)$.

The discrete-time Hamiltonian function can be defined as

$$H(x_k, h(x_k), \Delta V_k) = r(x_k, h(x_k)) + \gamma V^h(x_{k+1}) - V^h(x_k), \tag{57}$$

where $\Delta V_k = \gamma V_h(x_{k+1}) - V_h(x_k)$ is the forward difference operator. The Hamiltonian function captures the energy content along the trajectories of a system as reflected in the desired optimal performance. In fact, the Hamiltonian is the temporal difference error (49). The Bellman equation requires that the Hamiltonian be equal to zero for the value associated with a prescribed policy.

For the discrete-time linear quadratic regulator case, the system and Bellman equations are

$$x_{k+1} = Ax_k + Bu_k, \tag{58}$$

$$V^h(x_k) = \frac{1}{2} \sum_{i=k}^{\infty} \gamma^{i-k} (x_i^T Q x_i + u_i^T R u_i), \tag{59}$$

and the Bellman equation can be written in several ways as described in "Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation."

Policy Iteration and Value Iteration for Discrete-Time Dynamical Systems

Two forms of reinforcement learning can be based on policy iteration and value iteration. For temporal difference learning, policy iteration is written as follows in terms of the deterministic Bellman equation.

Policy Iteration Using Temporal Difference Learning

Initialize

Select some admissible control policy $h_o(x_k)$. Starting with $j = 0$, iterate on j until convergence:

Policy Evaluation

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1}). \tag{60}$$

Policy Improvement

$$h_{j+1}(x_k) = \arg\min_{h(\cdot)} (r(x_k, h(x_k)) + \gamma V_{j+1}(x_{k+1})), \tag{61}$$

or

$$h_{j+1}(x_k) = -\frac{\gamma}{2} R^{-1} g^T(x_k) \nabla V_{j+1}(x_{k+1}), \tag{62}$$

where $\nabla V(x) = \partial V(x)/\partial x$ is the gradient of the value function, interpreted here as a column vector.

Value iteration is similar but with the following policy evaluation procedure.

Value Iteration Using Temporal Difference Learning

Value Update Step. Update the value using

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1}). \tag{63}$$

In value iteration, any initial control policy $h_0(x_k)$ can be selected, which is not necessarily admissible or stabilizing.

"Bellman Equation for the Discrete-Time LQR, the Lyapunov Equation" shows that, for the discrete-time LQR, the Bellman equation (56) is a linear Lyapunov equation. "The Bellman Optimality Equation for Discrete-Time LQR Is an Algebraic Riccati Equation" shows that (54) yields the discrete-time algebraic Riccati equation (ARE). For the discrete-time LQR, the policy evaluation step (60) in policy iteration is a Lyapunov equation and policy iteration exactly corresponds to Hewer's algorithm [44] for solving the discrete-time ARE. Hewer proved that the algorithm converges under stabilizability and detectability assumptions. For the discrete-time LQR, value iteration is a Lyapunov recursion that converges to the solution to the discrete-time ARE under the stated assumptions by [45] (see "Policy Iteration and Value Iteration for the Discrete-Time LQR").

These policy iteration and value iteration algorithms are offline design methods that require knowledge of the discrete-time dynamics (A, B). The next section describes online methods for implementing policy iteration and value iteration that do not require full dynamics information.

Value Function Approximation

Policy iteration and value iteration can be implemented for a finite MDP by storing and updating lookup tables. The key to implementing policy and value iteration online for dynamical systems with infinite state and action spaces is to approximate the value function by a suitable approximator structure in terms of unknown parameters. Then, the unknown parameters are tuned online exactly as in system identification. This idea of *value function approximation* (VFA) is used by Werbos [14], [19] and called *approximate dynamic programming* (ADP) or *adaptive dynamic programming*. The approach is used by Bertsekas and Tsitsiklis [17] and called *neurodynamic programming* [6], [11].

For nonlinear systems (50), the value function contains higher order nonlinearities. We assume the Bellman equation (56) has a local smooth solution [47]. Then, according to the Weierstrass higher order approximation theorem, there exists a dense basis set $\{\varphi_i(x)\}$ such that

$$\begin{aligned} V(x) &= \sum_{i=1}^{\infty} w_i \varphi_i(x) \\ &= \sum_{i=1}^{L} w_i \varphi_i(x) + \sum_{i=L+1}^{\infty} w_i \varphi_i(x) \equiv W^T \phi(x) + \varepsilon_L(x), \end{aligned} \tag{64}$$

where basis vector $\phi(x) = [\varphi_1(x)\ \ \varphi_2(x) \cdots \varphi_L(x)]: R^n \to R^L$ and $\varepsilon_L(x)$ converges uniformly to zero as the number of terms retained $L \to \infty$. The standard usage of the Weierstrass theorem employs a polynomial basis set. In the neural network research, approximation results are shown for other basis sets including sigmoid, hyperbolic tangent, Gaussian radial basis functions, and others. There, standard results show that the neural network approximation error $\varepsilon_L(x)$ is bounded by a constant on a compact set, where L is the number of hidden-layer neurons, $\varphi_i(x)$ are the neural network activation functions, and w_i are the neural network weights.

In the LQR case, it is known that the value is quadratic in the state, so that $V(x_k) = (1/2) x_k^T P x_k$ for some kernel matrix P. Then the basis set $\{\varphi_i(x)\}$ consists of quadratic terms in the state components and the weight vector W consists of the elements of matrix P. Since P is symmetric and has only $n(n + 1)/2$ independent elements, there are $n(n + 1)/2$ independent elements in $\{\varphi_i(x)\}$.

Optimal Adaptive Control Algorithms for Discrete-Time Systems

With the above background, several adaptive control algorithms can be presented based on temporal difference reinforcement learning that converge online to the optimal control solution.

The parameters in $\tilde{p}$ or W are unknown. Substitution of the value function approximation $\hat{V}(x) = W^T\phi(x)$ into the value update (60) in policy iteration results in the following algorithm.

Optimal Adaptive Control Using a Policy Iteration Algorithm

Initialize

Select some admissible control policy $h_0(x_k)$. Starting with $j = 0$, iterate on j until convergence:

Policy Evaluation Step

Determine the least-squares solution W_{j+1} to

$$W_{j+1}^T(\phi(x_k) - \gamma\phi(x_{k+1})) = r(x_k, h_j(x_k)) = Q(x_k) + h_j^T(x_k)Rh_j(x_k). \quad (65)$$

Policy Improvement Step

Determine an improved policy using

$$h_{j+1}(x) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\nabla\phi^T(x_{k+1})W_{j+1}. \quad (66)$$

This algorithm is easily implemented online by standard system identification techniques [48]. Note that (65) is a scalar equation, whereas the unknown parameter vector $W_{j+1} \in R^L$ has L elements. Therefore, data from multiple time steps are needed for its solution. At time $k + 1$ we measure the previous state x_k, the control $u_k = h_j(x_k)$, the next state x_{k+1}, and compute the resulting utility $r(x_k, h_j(x_k))$. These data result in one scalar equation. This procedure is repeated for subsequent times using the same policy $h_j(\cdot)$ until at least L equations are obtained, at which point the least-squares solution W_{j+1} can be determined. Batch least-squares can be used for this procedure.

Alternatively, note that equations of the form (65) are exactly those solved by recursive least-squares (RLS) techniques [48]. Therefore, RLS can be run online until convergence. Write (65) as

$$W_{j+1}^T\Phi(k) \equiv W_{j+1}^T(\phi(x_k) - \gamma\phi(x_{k+1})) = r(x_k, h_j(x_k)), \quad (67)$$

with $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ being a regression vector. At step j of the policy iteration algorithm, the control policy is fixed at $u = h_j(x)$. Then, at each time k the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ is measured. One step of RLS is then performed. This procedure is repeated for subsequent times until convergence to the parameters corresponding to the value $V_{j+1}(x) = W_{j+1}^T\phi(x)$. For RLS to converge, the regression vector $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ must be persistently exciting.

An alternative to RLS is a gradient descent tuning method such as

$$W_{j+1}^{i+1} = W_{j+1}^i - \alpha\Phi(k)((W_{j+1}^i)^T\Phi(k) - r(x_k, h_j(x_k))), \quad (68)$$

with $\alpha > 0$ being a tuning parameter. The step index j is held fixed, and index i is incremented at each increment of the time index k. Note that the quantity inside the large brackets is just the temporal difference error.

Once the value parameters have converged, the control policy is updated according to (66). Then, the procedure is repeated for step $j + 1$. This entire procedure is repeated until convergence to the optimal control solution.

This method provides an online reinforcement learning algorithm for solving the optimal control problem using policy iteration by measuring data along the system trajectories. Likewise, an online reinforcement learning algorithm can be given based on value iteration. Substitution of the value function approximation into the value update (63) in value iteration results in the following algorithm.

Optimal Adaptive Control Using a Value Iteration Algorithm

Initialize

Select some control policy $h_0(x_k)$, not necessarily admissible or stabilizing. Starting with $j = 0$, iterate on j until convergence:

Value Update Step

Determine the least-squares solution W_{j+1} to

$$W_{j+1}^T\phi(x_k) = r(x_k, h_j(x_k)) + W_j^T\gamma\phi(x_{k+1}). \quad (69)$$

Policy Improvement Step

Determine an improved policy using (66).

Equation (69) can be solved in real time using batch least-squares, RLS, or gradient-based methods based on data $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ measured at each time along the system trajectories. Then the policy is improved using (66). Note that the old weight parameters are on the right-hand side of (69). Thus, the regression vector is now $\phi(x_k)$, which must be persistently exciting for convergence of RLS.

Introduction of a Second "Actor" Neural Network

Using value function approximation (VFA) allows standard system identification techniques to be used to find the value function parameters that approximately solve the Bellman equation. The approximator structure just described that is used for approximation of the value function is known as the *critic neural network*, as it

determines the value of using the current policy. Using VFA, the policy iteration reinforcement learning algorithm solves a Bellman equation during the value update portion of each iteration step j by observing only the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ at each time along the system trajectory and solving (65). In the case of value iteration, VFA is used to perform a value update using (69).

The critic network solves the Bellman equation using observed data without knowing the system dynamics. According to "Policy Iteration and Value Iteration for the Discrete-Time LQR," in the LQR case the critic solves Lyapunov equation (65) or Lyapunov recursions (69) without knowing the system matrices (A, B).

Note that in the LQR case the policy update (66) is given by

$$K^{j+1} = -(B^T P^{j+1} B + R)^{-1} B^T P^{j+1} A, \tag{70}$$

which requires full knowledge of the dynamics (A, B). Note further that the embodiment (66) cannot easily be implemented in the nonlinear case because it is implicit in the control, since x_{k+1} depends on $h(\cdot)$ and is the argument of a nonlinear activation function.

These problems are both solved by introducing a *second neural network* for the control policy, known as the *actor neural network* [14], [19], [34]. Consider a parametric approximator structure for the control action

$$u_k = h(x_k) = U^T \sigma(x_k), \tag{71}$$

with $\sigma(x): R^n \to R^M$ being a vector of M activation functions and $U \in R^{M \times m}$ being a matrix of weights or unknown parameters. In the LQR, the optimal state feedback is linear in the states so that the basis set $\sigma(x)$ can be taken as the state vector.

After convergence of the critic neural network parameters to W_{j+1} in policy iteration or value iteration, performing the policy update (66) is required. To achieve this aim, a gradient descent method for tuning the actor weights U such as

$$\begin{aligned} U_{j+1}^{i+1} = U_{j+1}^{i} - \beta \sigma(x_k)(2R(U_{j+1}^{i})^T \sigma(x_k) \\ + \gamma g(x_k)^T \nabla \phi^T(x_{k+1}) W_{j+1})^T \end{aligned} \tag{72}$$

can be used, with $\beta > 0$ being a tuning parameter [34]. The tuning index i can be incremented with the time index k.

The tuning of the actor neural network requires observations at each time k of the data set (x_k, x_{k+1}), that is, the current state and the next state. However, as per the formulation (71), the actor neural network yields the control u_k at time k in terms of the state x_k at time k. The next state x_{k+1} is not needed in (71). Thus, after (72) converges, (71) is a legitimate feedback controller. Note that, in the LQR case, the actor neural network (71) embodies the feedback gain computation (70). Equation (70) contains the state internal dynamics A, but (71) does not. Therefore, the A matrix is not needed to compute the feedback control. The reason is that the actor neural network learns information about A in its weights, since (x_k, x_{k+1}) are used in its tuning.

Finally, note that only the input function $g(\cdot)$ or, in the LQR case, the B matrix, is needed in (72) to tune the actor neural network. Introducing a second actor neural network completely avoids the need for knowledge of the state drift dynamics $f(\cdot)$, or the matrix A in the LQR case.

Example 1. Discrete-Time Optimal Adaptive Control of Power System Using Value Iteration

This example shows the use of discrete-time value iteration to solve the discrete-time ARE online without knowing the system matrix A. We simulate the online value iteration algorithm (69), (72) for load frequency control of an electric power system. Power systems are complicated nonlinear systems. However during normal operation the system load, which produces the nonlinearity, has only small variations. As such, a linear model can be used to represent the system dynamics around an operating point specified by a constant load value. A problem arises from the fact that in an actual plant the parameter values are not precisely known, reflected in an unknown system A matrix, yet an optimal control solution is sought.

The model of the system that is considered here is $\dot{x} = Ax + Bu$, where

$$A = \begin{bmatrix} -\frac{1}{T_P} & \frac{K_P}{T_P} & 0 & 0 \\ 0 & -\frac{1}{T_T} & -\frac{1}{T_T} & 0 \\ -\frac{1}{RT_G} & 0 & -\frac{1}{T_G} & -\frac{1}{T_G} \\ K_E & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{T_G} \\ 0 \end{bmatrix}.$$

The system state is $x(t) = [\Delta f(t)\ \Delta P_g(t)\ \Delta X_g(t)\ \Delta E(t)]^T$, where $\Delta f(t)$ is the incremental frequency deviation in hertz, $\Delta P_g(t)$ is the incremental change in the generator output (p.u. megawatt), $\Delta X_g(t)$ is the incremental change in the governor position in p.u. megawatt, and $\Delta E(t)$ is the incremental change in integral control. The system parameters are the governor time constant T_G, turbine time constant T_T, plant model time constant T_P, plant model gain K_P, speed regulation due to the governor action R, and the integral control gain K_E.

The values of the continuous-time system parameters were randomly picked within specified ranges so that

$$nA = \begin{bmatrix} -0.0665 & 8 & 0 & 0 \\ 0 & -3.663 & 3.663 & 0 \\ -6.86 & 0 & -13.736 & -13.736 \\ 0.6 & 0 & 0 & 0 \end{bmatrix},$$

$$B^T = [0 \quad 0 \quad 13.7355 \quad 0].$$

The discrete-time dynamics were obtained using the zero-order hold method with a sampling period of 0.01 s. The solution to the discrete-time ARE with cost function weights $Q = I_4$, $R = 1$, and $\gamma = 1$ is

$$P_{\text{DARE}} = \begin{bmatrix} 0.4805 & 0.4772 & 0.0604 & 0.4771 \\ 0.4772 & 0.7892 & 0.1240 & 0.3855 \\ 0.0604 & 0.1240 & 0.0567 & 0.0304 \\ 0.4771 & 0.3855 & 0.0304 & 2.3513 \end{bmatrix}.$$

In this simulation, only the time constant T_G of the governor, which appears in the B matrix, is considered to be known, while the values for all the other parameters appearing in the system A matrix are not known. That is, the A matrix is needed only to simulate the system and obtain the data and is not needed by the control algorithm.

For the discrete-time LQR, the value is quadratic in the states, $V(x) = (1/2)x^T Px$. Therefore, the basis functions for the critic neural network in (64) are selected as the quadratic polynomial vector in the state components. Since there are $n = 4$ states, this vector has $n(n+1)/2 = 10$ components. The control is linear in the states, $u = -Kx$, and the basis functions for the actor neural network (71) are taken as the state components.

Value iteration can be implemented online by setting up a batch least-squares problem to solve for the ten critic neural network parameters, which are in the Riccati solution entries W_{j+1} in (69) for each step j. In this simulation, the matrix P^{j+1} corresponding to W_{j+1} is determined after collecting 15 points of data $(x_k, x_{k+1}, r(x_k, u_k))$ for each least-squares problem. This least-squares problem for the critic weights is solved each 0.15 s. Then the actor neural network parameters, that is, the feedback gain matrix entries, are updated using (72). The simulations were performed over a time interval of 60 s.

The system state trajectories are shown in Figure 3, which shows that the states are regulated to zero as desired. The convergence of the Riccati matrix parameters is shown in Figure 4. The final values of the critic neural network parameter estimates are

$$P_{\text{critic NN}} = \begin{bmatrix} 0.4802 & 0.4768 & 0.0603 & 0.4754 \\ 0.4768 & 0.7887 & 0.1239 & 0.3834 \\ 0.0603 & 0.1239 & 0.0567 & 0.0300 \\ 0.4754 & 0.3843 & 0.0300 & 2.3433 \end{bmatrix}.$$

The optimal adaptive control value iteration algorithm converges to the optimal control solution as given by the algebraic Riccati equation solution. This solution is performed in real time without knowing the system A matrix.

Actor-Critic Implementation of Discrete-Time Optimal Adaptive Control

Two algorithms for optimal adaptive control of discrete-time systems based on reinforcement learning have given a policy iteration algorithm implemented by solving (65) using RLS and a policy update by running (72) and a value iteration algorithm implemented by solving (69) using RLS and a policy update by running (72).

The implementation of reinforcement learning using two neural networks, one as a critic and one as an actor, yields the actor/critic reinforcement learning structure shown in Figure 1. In this control system, the critic and the actor are tuned online using the observed data $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ along the system trajectory. The critic and actor are tuned sequentially in both the policy iteration and the value iteration algorithms. That is, the

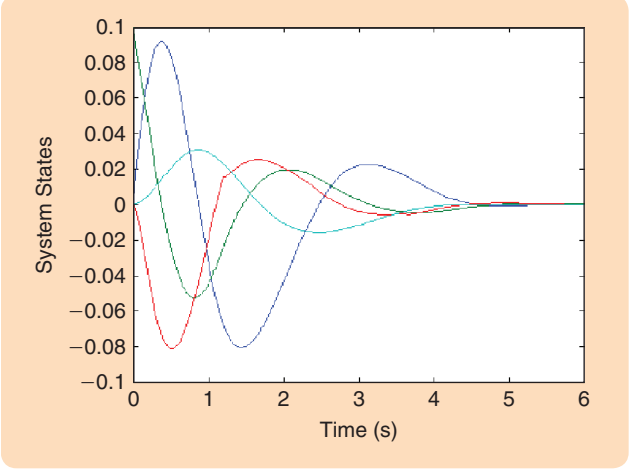

FIGURE 3 System states during the first 6 s. This figure shows that, even though the A matrix of the power system is unknown, the adaptive controller based on value iteration keeps the states stable and regulates them to zero.

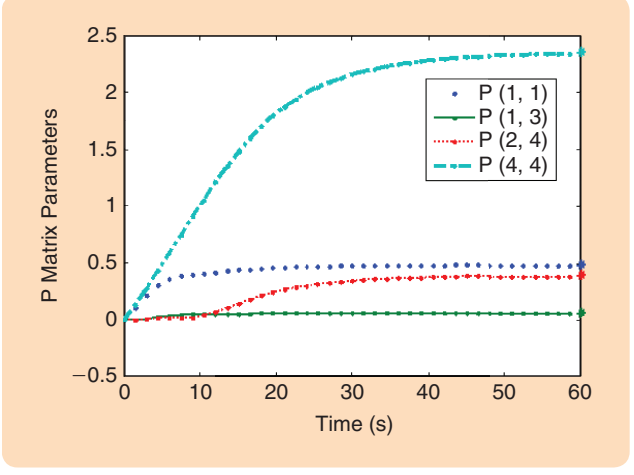

FIGURE 4 Convergence of selected algebraic Riccati equation solution parameters. This plot shows that the adaptive controller based on value iteration converges to the ARE solution in real time without knowing the system matrix A.

Adaptive Controller for Online Solution of Discrete-Time LQR Using Q Learning

This sidebar presents an adaptive control algorithm based on Q learning that converges online to the solution to the discrete-time LQR problem. This is accomplished by solving the algebraic Riccati equation in real time without knowing the system dynamics by using data measured along the system trajectories.

Q learning is implemented by repeatedly performing the iterations (79) and (80). In it is seen that the LQR Q function is quadratic in the states and inputs so that $Q(x_k,u_k) = Q(z_k) \equiv (1/2) z_k^T S z_k$ where $z_k = [x_k^T \; u_k^T]^T$. The kernel matrix S is explicitly given by (S22) in terms of the system parameters A and B. However, matrix S can be estimated online without knowing A and B by using system identification techniques. Specifically, write the Q function in parametric form as

$$Q(x,u) = Q(z) = W^T \phi(z), \tag{S26}$$

with W being the vector of the elements of S and the basis vector $\phi(z)$ consisting of quadratic terms in the elements of z, which contains state and input components. Redundant entries are removed so that W is composed of the $(n+m)(n+m+1)/2$ elements in the upper portion of S, with $x_k \in R^n, u_k \in R^m$.

Now, for the LQR, the Q learning Bellman equation (79) can be written as

$$W_{j+1}^T (\phi(z_k) - \phi(z_{k+1})) = \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k). \tag{S27}$$

Note that the Q matrix here is the state weighting matrix in the performance index; it should not be confused with the Q function $Q(x_k,u_k)$. This equation must be solved at each step j of the Q learning process. Note that (S27) is one equation in $n(n+1)/2$ unknowns, namely the entries of vector W. This is exactly the sort of equation encountered in system identification, and is solved online using methods from adaptive control such as recursive least squares (RLS).

Therefore, Q learning is implemented as follows.

INITIALIZE.

Select an initial feedback policy $u_k = -K^0 x_k$ at $j = 0$. The initial gain matrix need not be stabilizing and can be selected equal to zero.

STEP j.

Identify the Q Function Using RLS.

At time k, apply the control u_k based on the current policy $u_k = -K^j x_k$ and measure the data set $(x_k, u_k, x_{k+1}, u_{k+1})$, where u_{k+1} is computed using $u_{k+1} = -K^j x_{k+1}$. Compute the quadratic basis sets $\phi(z_k), \phi(z_{k+1})$. Now perform a one-step update in the parameter vector W by applying RLS to equation (S27). Repeat at the next time $k+1$ and continue until RLS converges and the new parameter vector W_{j+1} is found.

Update the Control Policy

Unpack the vector W_{j+1} into the kernel matrix

$$Q(x_k,u_k) \equiv \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T S \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} S_{xx} & S_{xu} \\ S_{ux} & S_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \tag{S28}$$

Perform the control update using (S24), which is

$$u_k = -S_{uu}^{-1} S_{ux} x_k, \tag{S29}$$

SET $j = j+1$. GO TO STEP j.

TERMINATION.

This algorithm is terminated when there are no further updates to the Q function or the control policy at each step.

This is an adaptive control algorithm implemented using Q function identification by RLS techniques. No knowledge of the system dynamics (A, B) is needed for its implementation. The algorithm effectively solves the algebraic Riccati equation online in real-time using data $(x_k, u_k, x_{k+1}, u_{k+1})$ measured in real time at each time stage k. It is necessary to add probing noise to the control input to guarantee persistence of excitation to solve (S27) using RLS.

weights of one neural network are held constant while the weights of the other are tuned until convergence. This procedure is repeated until both neural networks have converged. Thus, the controller learns the optimal controller online. This procedure amounts to an *online adaptive optimal control system* wherein the value function parameters are tuned online and the convergence is to the optimal value and control. The convergence of value iteration using two neural networks for the discrete-time nonlinear system (50) is proven in [34].

According to reinforcement learning principles, the optimal adaptive control structure requires two loops, a critic and an actor, and operates at multiple timescales. A fast loop implements the control action inner loop, a slower timescale operates in the critic loop, and a third timescale operates to update the policy.

Q Learning for Optimal Adaptive Control

The above text described how to implement an optimal adaptive controller using reinforcement learning that only requires knowledge of the system input function $g(x_k)$. The Q learning reinforcement learning method results in an adaptive control algorithm that converges online to the optimal control solution for completely unknown systems. That is, it solves the Bellman equation (56) and the HJB equation (54) online in real time by using data measured along the system trajectories, without any knowledge of the dynamics $f(x_k)$, $g(x_k)$.

A method known as Q learning allows the learning of optimal control solutions online, in the discrete-time case, for completely unknown systems.

Q learning, developed by Watkins [42], [43] and Werbos [7], [14], [19], is a simple method for reinforcement learning that works for unknown MDPs, that is for systems with completely unknown dynamics. Q learning is called *action-dependent heuristic dynamic programming* (ADHDP) by Werbos, since the Q function depends on the control input. Q learning learns the Q function (37) using temporal difference methods by performing an action u_k and measuring at each time stage the resulting data experience set (x_k, x_{k+1}, r_k) consisting of the current state, the next state, and the resulting stage cost. Writing the Q function Bellman equation (39) along a sample path gives

$$Q^\pi(x_k, u_k) = r(x_k, u_k) + \gamma Q^\pi(x_{k+1}, h(x_{k+1})), \tag{73}$$

which defines a temporal difference error

$$e_k = -Q^\pi(x_k, u_k) + r(x_k, u_k) + \gamma Q^\pi(x_{k+1}, h(x_{k+1})). \tag{74}$$

The value iteration algorithm for Q function is given by (46). Based on this, the Q function is updated using the algorithm

$$\begin{aligned} Q_k(x_k,u_k) = Q_{k-1}(x_k,u_k) + \alpha_k[r(x_k,u_k) \\ + \gamma \min_u Q_{k-1}(x_{k+1},u) - Q_{k-1}(x_k,u_k)]. \end{aligned} \tag{75}$$

This algorithm is developed for finite MDPs and its convergence has been proven by Watkins [42] using stochastic approximation (SA) methods. It is shown the algorithm converges for a finite MDP provided that all state-action pairs are visited infinitely often and

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum \alpha_k^2 < \infty, \tag{76}$$

which are standard stochastic approximation conditions. On convergence, the temporal difference error is approximately equal to zero. For a finite MDP, Q learning requires storing a lookup table in terms of all the states x and actions u.

The requirement that all state-action pairs are visited infinitely often translates to the problem of maintaining sufficient *exploration* during learning.

The Q learning algorithm (75) is similar to SA methods for adaptive control or parameter estimation used in the control systems literature. Below is a derivation of methods for Q learning for dynamical systems that yield adaptive control algorithms that converge to optimal control solutions.

Policy iteration and value iteration algorithms can be given using the Q function in (42)–(46). A Q learning algorithm is easily developed for discrete-time dynamical systems using Q function approximation [7], [14], [19], [49]. It is shown in "Q Function for the Discrete-Time LQR" that, for the discrete-time LQR, the Q function is a quadratic form in terms of $z_k \equiv [x_k^T \; u_k^T]^T$. Assume that, for nonlinear systems, the Q function is parameterized as

$$Q(x, u) = W^T \phi(z), \tag{77}$$

for some unknown parameter vector W and basis set vector $\phi(z)$. Substituting the Q function approximation into the temporal difference error (74) yields

$$e_k = -W^T \phi(z_k) + r(x_k, u_k) + \gamma W^T \phi(z_{k+1}), \tag{78}$$

on which either policy iteration or value iteration algorithms can be based. Considering the policy iteration algorithm (42), equation (43) yields the Q function evaluation step

$$W_{j+1}^T(\phi(z_k) - \gamma\phi(z_{k+1})) = r(x_k, h_j(x_k)), \tag{79}$$

and the policy improvement step

$$h_{j+1}(x_k) = \arg\min_u (W_{j+1}^T \phi(x_k, u)), \text{ for all } x \in X. \tag{80}$$

Q learning using value iteration (44) is given by

$$W_{j+1}^T \phi(z_k) = r(x_k, h_j(x_k)) + \gamma W_j^T \phi(z_{k+1}), \tag{81}$$

and (80). These equations do not require knowledge of the dynamics $f(\cdot), g(\cdot)$.

For online implementation, batch least-squares or RLS can be used to solve (79) for the parameter vector W_{j+1} given the regression vector $(\phi(z_k) - \gamma\phi(z_{k+1}))$, or (81) using regression vector $\phi(z_k)$. The observed data at each time instant are $(z_k, z_{k+1}, r(x_k, u_k))$ with $z_k \equiv [x_k^T u_k^T]^T$. The vector $z_{k+1} \equiv [x_{k+1}^T u_{k+1}^T]^T$ is computed using $u_{k+1} = h_j(x_{k+1})$

with $h_j(\cdot)$ being the current policy. Probing noise must be added to the control input to obtain persistence of excitation. On convergence, the action update (80) is performed. This update is easily accomplished without knowing the system dynamics due to the fact that the Q function contains u_k as an argument, therefore $\partial(W_{j+1}^T\phi(x_k,u))/\partial u$ can be explicitly computed. The details are presented in.

Due to the simple form of the action update (80), the actor neural network is not needed for Q learning; it can be implemented using only one neural network for Q function approximation.

Approximate Dynamic Programming Using Output Feedback

The aforementioned methods have relied on full state variable feedback. Little work has been done in applications of reinforcement learning for feedback control using output feedback, which corresponds to partially observable Markov processes. Design of an ADP controller that uses only output feedback is given in [50].

INTEGRAL REINFORCEMENT LEARNING FOR OPTIMAL ADAPTIVE CONTROL OF CONTINUOUS-TIME SYSTEMS

Reinforcement learning is considerably more difficult for continuous-time systems than for discrete-time systems, and fewer results are available. The development of an offline policy iteration method for continuous-time systems is described in [51]. Using a method known as *integral reinforcement learning* (IRL) [37], [15] allows the application of reinforcement learning to formulate online optimal adaptive control methods for continuous-time systems. These methods find solutions to optimal HJ design equations and Riccati equations online in real time without knowing the system drift dynamics $f(x)$, or in the LQR case, without knowing the A matrix.

Consider the continuous-time nonlinear dynamical system

$$\dot{x} = f(x) + g(x)u, \tag{82}$$

with state $x(t) \in R^n$, control input $u(t) \in R^m$, an equilibrium point at $x = 0$, e.g., $f(0) = 0$, and $f(x) + g(x)u$ Lipschitz on a set $\Omega \subseteq R^n$ that contains the origin. Assume that the system is stabilizable on Ω, that is, there exists a continuous control function $u(t)$ such that the closed-loop system is asymptotically stable on Ω.

Define a performance measure or cost function that has the value associated with the feedback control policy $u = \mu(x)$ given by

$$V^{\mu}(x(t)) = \int_t^{\infty} r(x(\tau), u(\tau))d\tau, \tag{83}$$

with utility $r(x,u) = Q(x) + u^T Ru$, positive-definite $Q(x)$, that is, $Q(x) > 0$ for all x and $x = 0 \Rightarrow Q(x) = 0$, and positive-definite matrix $R = R^T \in R^{m\times m}$.

For the continuous-time LQR, the above expressions are

$$\dot{x} = Ax + Bu, \tag{84}$$

$$V^{\mu}(x(t)) = \frac{1}{2}\int_t^{\infty} (x^T Qx + u^T Ru)d\tau. \tag{85}$$

A policy is called *admissible* if it is continuous, stabilizes the system, and has a finite associated cost. If the cost is smooth, then an infinitesimal equivalent to (83) can be found by Leibniz's formula to be

$$0 = r(x,\mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)), V^{\mu}(0) = 0, \tag{86}$$

where ∇V^{μ}, taken here as a column vector, denotes the gradient vector of the cost function V^{μ} with respect to x.

Equation (86) is the continuous-time Bellman equation. This equation is defined based on the continuous-time Hamiltonian function

$$H(x,\mu(x),\nabla V^{\mu}) = r(x,\mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)). \tag{87}$$

The optimal value satisfies the continuous-time (HJB) equation [3]

$$0 = \min_{\mu} H(x,\mu(x),\nabla V^*) \tag{88}$$

and the optimal control satisfies

$$\mu^* = \arg\min_{\mu} H(x,\mu(x),\nabla V^*). \tag{89}$$

These expressions show why it is much more challenging to apply reinforcement learning to continuous-time systems. Comparing the continuous-time Hamiltonian (87) to the discrete-time Hamiltonian (57), it is seen that (87) contains the full system dynamics $f(x) + g(u)u$, whereas (57) does not. What this means is that the continuous-time Bellman equation (86) cannot be used as a basis for reinforcement learning unless the full dynamics are known.

Reinforcement learning methods based on (86) can be developed [52], [29], [40]. These methods have limited use for adaptive control purposes because the system dynamics must be known, state derivatives must be measured, or integration over an infinite horizon is required. In another approach, Euler's method can be used to discretize the continuous-time Bellman equation (86) (see [52]). Noting that

$$\begin{aligned} 0 &= r(x,\mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)) \\ &= r(x,\mu(x)) + \dot{V}^{\mu}, \end{aligned} \tag{90}$$

The optimal adaptive controllers presented in this article are a natural extension of adaptive controllers.

Euler's forward method can be used to discretize this partial differential equation to obtain

$$0 = r(x_k, u_k) + \frac{V^\mu(x_{k+1}) - V^\mu(x_k)}{T} \equiv \frac{r_S(x_k, u_k)}{T} + \frac{V^\mu(x_{k+1}) - V^\mu(x_k)}{T}, \quad (91)$$

with sample period T so that $t = kT$. The discrete sampled utility is $r_S(x_k, u_k) = r(x_k, u_k)T$.

Now note that the discretized continuous-time Bellman equation (91) has the same form as the discrete-time Bellman equation (56). Therefore, all the reinforcement learning methods just described for discrete-time systems can be applied.

However, (91) is an approximation only. An alternative exact method for continuous-time reinforcement learning is given in [37], [15]. That method is termed IRL. Note that the cost (83) can be written in the *integral reinforcement form*

$$V^\mu(x(t)) = \int_t^{t+T} r(x(\tau), u(\tau))d\tau + V^\mu(x(t+T)), \quad (92)$$

for some $T > 0$. This equation is exactly in the form of the discrete-time Bellman equation (56). According to Bellman's principle, the optimal value is given in terms of this construction as [3]

$$V^*(x(t)) = \min_{\bar{u}(t:t+T)} \left(\int_t^{t+T} r(x(\tau), u(\tau))d\tau + V^*(x(t+T)) \right),$$

where $\bar{u}(t:t+T) = \{u(\tau): t \le \tau < t+T\}$. The optimal control is

$$\mu^*(x(t)) = \arg\min_{\bar{u}(t:t+T)} \left(\int_t^{t+T} r(x(\tau), u(\tau))d\tau + V^*(x(t+T)) \right).$$

It is shown in [37] that the Bellman equation (86) is equivalent to the integral reinforcement form (92). That is, the positive-definite solution of both equations is the value (83) of the policy $u = \mu(x)$.

The integral reinforcement form (92) serves as a Bellman equation for continuous-time systems and serves as a fixed point equation. Therefore, the temporal difference error for continuous-time systems can be defined as

$$e(t:t+T) = \int_t^{t+T} r(x(\tau), u(\tau))d\tau + V^\mu(x(t+T)) - V^\mu(x(t)). \quad (93)$$

This equation does not involve the system dynamics.

It is straightforward to use the above equation to formulate policy iteration and value iteration for continuous-time systems. The following algorithms are termed as *integral reinforcement learning for continuous-time systems* [37], [15]. Both algorithms give optimal adaptive controllers for continuous-time systems, that is, adaptive control algorithms that converge to optimal control solutions.

IRL Optimal Adaptive Control Using Policy Iteration

Initialize

Select some admissible control policy $\mu_0(x)$. Starting with $j = 0$, iterate on j until convergence:

Policy Evaluation Step

Solve for $V_{j+1}(x(t))$ using

$$V_{j+1}(x(t)) = \int_t^{t+T} r(x(s), \mu_j(x(s)))ds + V_{j+1}(x(t+T)) \quad \text{with } V_{j+1}(0) = 0. \quad (94)$$

Policy Improvement Step

Determine an improved policy using

$$\mu_{j+1} = \arg\min_u [H(x, u, \nabla V_{j+1})], \quad (95)$$

which explicitly is

$$\mu_{j+1}(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla V_{j+1}. \quad (96)$$

IRL Optimal Adaptive Control Using Value Iteration

Initialize

Select some control policy $\mu_0(x)$, which is not necessarily stabilizing. Starting with $j = 0$, iterate on j until convergence.

Policy Evaluation Step

Solve for $V_{j+1}(x(t))$ using

$$V_{(j+1)}(x(t)) = \int_t^{t+T} r(x(s), \mu_j(x(s)))ds + V_j(x(t+T)). \quad (97)$$

Policy Improvement Step

Determine an improved policy using (96).

Neither algorithm requires knowledge about the system drift dynamics function $f(x)$. That is, the algorithms are applicable to partially unknown systems. Convergence of IRL policy iteration is proved in [37].

Online Implementation of IRL: A Hybrid Optimal Adaptive Controller

Both of these IRL algorithms can be implemented online by reinforcement learning techniques using value function approximation $V(x) = W^T\phi(x)$ in a critic approximator network. Using VFA in the policy iteration algorithm (94) yields

$$W_{j+1}^T[\phi(x(t)) - \phi(x(t+T))] = \int_t^{t+T} r(x(s), \mu_j(x(s)))ds. \quad (98)$$

Using VFA in the value iteration algorithm (97) yields

$$W_{j+1}^T\phi(x(t)) = \int_t^{t+T} r(x(s), \mu_j(x(s)))ds + W_j^T\phi(x(t+T)). \quad (99)$$

Then RLS or batch least-squares can be used to update the value function parameters in these equations. On convergence of the value parameters, the action is updated using (96).

IRL provides an optimal adaptive controller, that is, an adaptive controller that measures data along the system trajectories and converges to optimal control solutions. Only the system input coupling dynamics $g(x)$ is needed to implement these algorithms. The drift dynamics $f(x)$ is not needed.

The implementation of IRL optimal adaptive control is shown in Figure 5. The time is incremented at each iteration by the period T. The reinforcement learning time interval T need not be the same at each iteration. The value of T can be changed depending on how long it takes to receive meaningful information from the observations; T is not a sample period in the standard meaning.

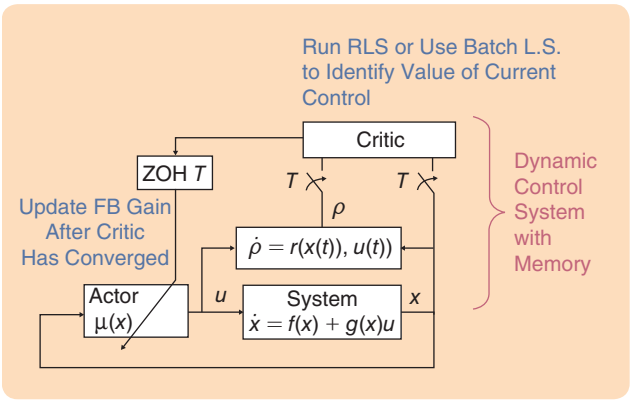

FIGURE 5 Hybrid optimal adaptive controller based on integral reinforcement learning (IRL), which shows the two-time scale hybrid nature of the IRL controller. The integral reinforcement signal is added as an extra state and functions as the memory of the controller. The Critic runs on a slow time scale and learns the value of using the current control policy. When the Critic converges, the Actor control policy is updated to obtain an improved value.

The measured data at each time increment are $(x(t), x(t+T), \rho(t{:}t+T))$ where

$$\rho(t{:}t+T) = \int_t^{t+T} r(x(\tau), u(\tau))d\tau \quad (100)$$

is the *integral reinforcement* measured on each time interval. The integral reinforcement can be computed in real time by introducing an integrator $\dot{\rho} = r(x(t), u(t))$ as shown in Figure 5. That is, the integral reinforcement $\rho(t)$ is added as an extra continuous-time state that functions as the memory or controller dynamics. The remainder of the controller is a sampled-data controller.

The control policy $\mu(x)$ is updated periodically after the critic weights have converged to the solution to (98) or (99). Therefore, the policy is piecewise constant in time. On the other hand, the control varies continuously with the state between each policy update. It is seen that IRL for continuous-time systems is in fact a hybrid continuous-time/discrete-time adaptive controller that converges to the optimal control solution in real time without knowing the drift dynamics $f(x)$. The optimal adaptive controller has multiple control loops and several timescales. The inner control action loop operates in continuous-time. Data are sampled at intervals of length T. The critic network operates at a slower timescale depending on how long it takes to solve (98) or (99).

Due to the fact that the policy update (96) for continuous-time systems does not involve the drift dynamics $f(x)$, no actor neural network is needed in IRL. Only a critic neural network is needed for VFA.

Online Solution of Algebraic Riccati Equation Without Full Plant Dynamics

It can be shown that the integral reinforcement form (92) is equivalent to the Bellman equation (86) [37]. The IRL controller solves the Bellman equation online without knowing the drift dynamics $f(x)$. Moreover, IRL converges to the optimal control so that it solves the HJB equation (88).

In the continuous-time LQR case (84), (85) the control policies are linear state feedbacks $u = -Kx$. Then the Bellman equation (86) is the Lyapunov equation

$$(A - BK)^T P + P(A - BK) + Q + K^T RK = 0, \quad (101)$$

and (88) becomes the continuous-time ARE

$$A^T P + PA + Q - PBR^{-1}B^T P = 0. \quad (102)$$

Adaptive controllers do not generally converge to optimal solutions, and optimal controllers are designed offline using full dynamics information by solving matrix design equations.

Thus, IRL solves both the Lyapunov equation and the ARE online in real time, using data measured along the system trajectories, without knowing the A matrix.

For the continuous-time LQR, (94) is equivalent to a Lyapunov equation at each step, so that policy iteration is exactly the same as Kleinman's algorithm [55] for solving the continuous-time Riccati equation. This algorithm is a Newton's method for finding the optimal value. Continuous-time value iteration, on the other hand, is an algorithm that solves the continuous-time ARE based on iterations on certain discrete-time Lyapunov equations that are equivalent to (97).

Example 2. Continuous-Time Optimal Adaptive Control Using IRL

This example shows the hybrid control nature of the IRL optimal adaptive controller. It is also shown that IRL finds the solution to the ARE online without solving the ARE and without knowing the system matrix A. Consider the simple dc motor model

$$\dot{x} = Ax + Bu = \begin{bmatrix} -10 & 1 \\ -0.002 & -2 \end{bmatrix} x + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u,$$

with cost weight matrices $Q = I$, $R = 1$. The solution to the continuous-time ARE is computed to be

$$P = \begin{bmatrix} 0.05 & 0.0039 \\ 0.0039 & 0.2085 \end{bmatrix}.$$

This simulation uses the IRL-based continuous-time value iteration algorithm. This algorithm does not require knowledge of the system matrix A. For the continuous-time LQR, the value is quadratic in the states and the basis functions for the critic neural network are selected as the quadratic polynomial vector in the state components, $\phi(x) = [x_1^2 \;\; x_1x_2 \;\; x_2^2]$. The IRL time interval was selected as $T = 0.04$ s. A batch least-squares solution was used to update the three critic weights W_{j+1}, that is, the ARE solution elements, using (99). Measurements of the data set $(x(t), x(t+T), \rho(t{:}t+T))$ are taken over three time intervals of $T = 0.04$ s. Then, provided that there is enough excitation in the system, after each 0.12 s enough data are collected from the system to solve for the value of the matrix P. Then a greedy policy update is performed using (96), that is $u = -R^{-1}B^TPx \equiv -Kx$.

The state trajectories in Figure 6 show that suitable regulation is achieved. The control input and feedback gains are shown in Figure 7. The control gains are piecewise constant, while the control input is a continuous function of the state between policy updates. The critic neural network parameter estimates are shown in Figure 8. They converge almost exactly to the entries in the Riccati solution matrix P. Thus, the ARE is solved online without knowing the system matrix A.

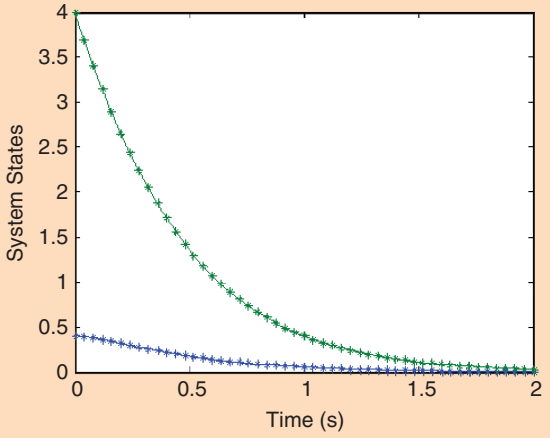

FIGURE 6 System states during the first 2 s, which shows that the continuous-time IRL adaptive controller regulates the states to zero without knowing the system matrix A.

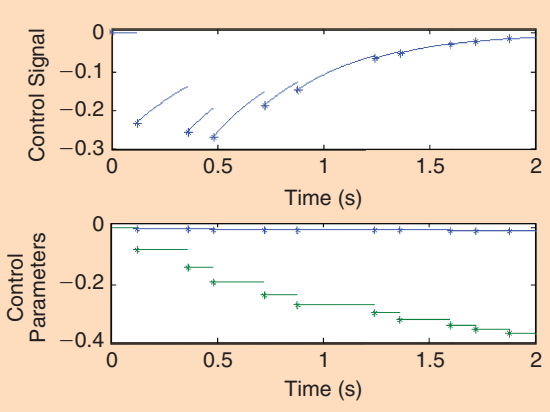

FIGURE 7 Control input and feedback gains, which show the hybrid nature of the IRL optimal adaptive controller. The controller gain parameters are discontinuous and piecewise constant, while the control signal itself is continuous between the gain parameter updates.

Example 3. Continuous-Time Optimal Adaptive Control for Power System Using IRL

This example shows that IRL finds the solution to the ARE online without solving the ARE and without knowing the system matrix A. Here the continuous-time IRL optimal adaptive control for the electric power system in Example 1 is demonstrated. The same system matrices and performance index are used as in Example 1. The solution to the continuous-time ARE is computed to be

$$P_{\text{ARE}} = \begin{bmatrix} 0.4750 & 0.4766 & 0.0601 & 0.4751 \\ 0.4766 & 0.7831 & 0.1237 & 0.3829 \\ 0.0601 & 0.1237 & 0.0513 & 0.0298 \\ 0.4751 & 0.3829 & 0.0298 & 2.3370 \end{bmatrix}.$$

This matrix is close to the discrete-time ARE solution presented in Example 1 since the sample period used there is small.

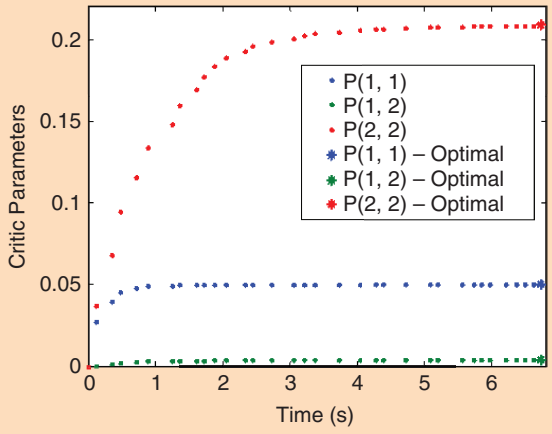

FIGURE 8 *P* matrix parameter estimates, which shows that the IRL adaptive controller converges online to the optimal Riccati equation solution without knowing the system matrix *A*.

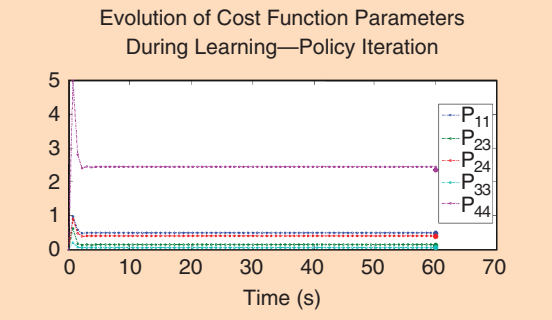

FIGURE 9 *P* matrix parameter estimates for IRL policy iteration, which shows that the IRL adaptive controller converges online to the optimal Riccati equation solution without knowing the system matrix *A*.

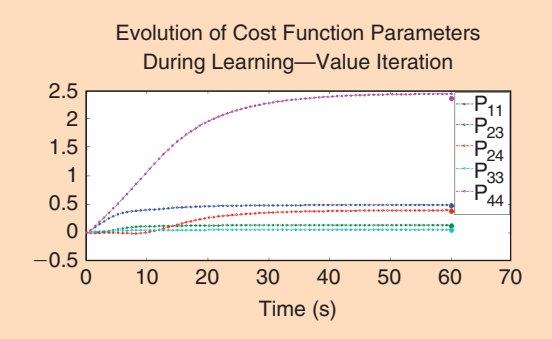

FIGURE 10 *P* matrix parameter estimates for IRL value iteration, which shows that the IRL adaptive controller converges online to the optimal Riccati equation solution without knowing the system matrix *A*.

The IRL policy iteration algorithm and the IRL value iteration algorithm are simulated. Neither requires knowledge of the system matrix A. The IRL time interval was taken as $T = 0.05$ s. Note that the IRL interval is not related at all to the sample period used to discretize the system in Example 1. It need not even be taken as a fixed constant for each step. Fifteen data points $(x(t), x(t+T), \rho(t{:}t+T))$ were taken to compute each batch least-squares update for the critic parameters $\bar{p}_{j+1} \equiv W_{j+1}$, that is, the elements of the ARE solution P, using (99). The value estimate was updated every 0.75 s. Then, the policy was computed using (96), that is, $u = -R^{-1}B^T Px \equiv -Kx$.

The state trajectories are similar to those presented in Example 1 and so are not plotted here. In the figures showing the value function parameter convergence, the star symbols represent the true values of the parameters of the optimal cost function calculated by solving the continuous-time ARE. For learning using the IRL policy iteration algorithm, the critic parameter estimates for the entries of the P matrix are shown in Figure 9. For learning using the IRL value iteration algorithm, the critic parameter estimates for the P matrix entries are shown in Figure 10. In both cases, the parameters converge to the true solution to the continuous-time ARE. Thus, the ARE is solved online without knowing the system A matrix. Note that IRL policy iteration converges far faster than IRL value iteration. Policy iteration is a Newton-Raphson method, so that its convergence rate is quadratic.

Note that far less computation is needed using this IRL algorithm on the continuous-time dynamics than is used in Example 1 for the discrete-time optimal adaptive control algorithm. There, the critic parameter estimates were updated every 0.15 s. Yet, the parameter estimates for the P matrix entries in Figures 4 and 10 almost overlay each other.

Now let's simulate the effects of changes in the A matrix. At $t = 30$ s, the A matrix was changed to

$$A_{(t \geq 30\,s)} = \begin{bmatrix} -0.0665 & 10 & 0 & 0 \\ 0 & -5.663 & 5.663 & 0 \\ -6.86 & 0 & -13.736 & -13.736 \\ 0.6 & 0 & 0 & 0 \end{bmatrix}.$$

According to the discussion in Example 1, this corresponds to a change in K_p/T_p from eight to ten and of $1/T_T$ from 3.663 to 5.663. The parameters K_P, T_P, T_T are subject to change on variations in the operating point of the machine, such as what occurs on load changes. For learning using the IRL policy iteration algorithm, the resulting critic parameter estimates for the P matrix entries are shown in Figure 11. For learning using the IRL value iteration algorithm, the resulting critic parameter estimates for the P matrix entries are shown in Figure 12. In both cases, the parameters converge after 30 s to the new optimal solution of the ARE corresponding to the new A matrix. This is accomplished without knowing either the old or the new A matrix.

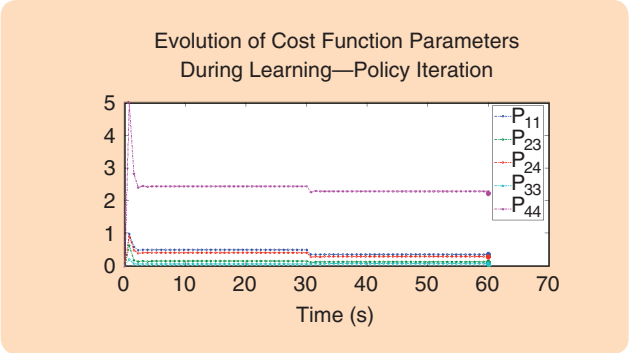

FIGURE 11 P matrix parameter estimates for IRL policy iteration with a change in the A matrix at 30 s, which shows that the IRL adaptive controller converges after 30 s to the new optimal Riccati equation solution corresponding to the new A matrix.

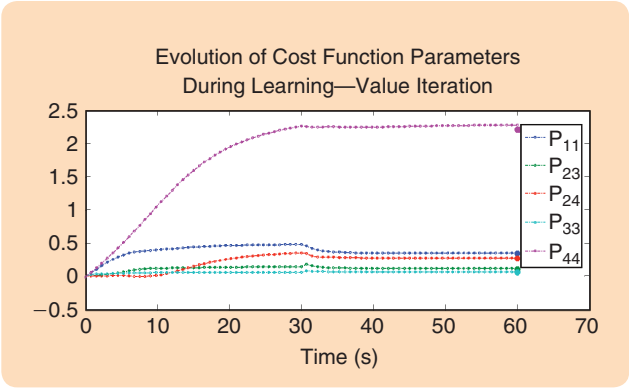

FIGURE 12 P matrix parameter estimates for IRL value iteration with a change in the A matrix at 30 s, which shows that the IRL adaptive controller converges after 30 s to the new optimal Riccati equation solution corresponding to the new A matrix.

Example 4. Continuous-Time IRL Optimal Adaptive Control for Nonlinear System

This example demonstrates the use of IRL to solve the HJB equation for nonlinear continuous-time systems by using data measured along the trajectories in real time. This example was developed using the converse HJB approach [56], which allows construction of nonlinear systems starting from the known optimal cost function.

Consider the nonlinear system given by

$$\begin{cases} \dot{x}_1 = -x_1 + x_2 + 2x_2^3 \\ \dot{x}_2 = f(x) + g(x)u \end{cases},$$

with $f(x) = -0.5(x_1 + x_2) + 0.5x_2(1 + 2x_2^2)\sin^2 x_1$, $g(x) = \sin x_1$. With the definitions $Q(x) = x_1^2 + x_2^2 + 2x_2^4$, $R = 1$, then the optimal cost function for this system is $V^*(x) = 0.5x_1^2 + x_2^2 + x_2^4$ and the optimal controller is $u^*(x) = -(x_2 + 2x_2^3)\sin x_1$. It can be verified that the HJB equation and the Bellman equation are both satisfied for these choices.

The cost function is approximated by the smooth function $V_j(x(t)) = W_j^T \phi(x(t))$ with $L = 8$ neurons and $\phi(x) = [x_1^2 \;\; x_1x_2 \;\; x_2^2 \;\; x_1^4 \;\; x_1^3x_2 \;\; x_1^2x_2^2 \;\; x_1x_2^3 \;\; x_2^4]^T$. The policy iteration IRL algorithm (98), (96) was used.

To ensure exploration so that the HJB solution is found over a suitable region, data were taken along five trajectories defined by five different initial conditions chosen randomly in the region $\Omega = \{-1 \le x_i \le 1; i = 1,2\}$. The IRL time period was taken as $T = 0.1$ s. At each iteration step, a batch least-squares problem was used to solve for the eight neural network weights using 40 data points measured on each of the five trajectories in Ω. Each data point consisted of $(x(t), x(t+T), \rho(t{:}t+T))$ with $\rho(t{:}t+T)$ the measured integral reinforcement cost. In this way, at every 4 s, the value was computed and then a policy update was performed.

The result of applying the algorithm is presented in Figure 13, which shows that the parameters of the critic neural network converged to the coefficients of the optimal cost function $V^*(x) = 0.5x_1^2 + x_2^2 + x_2^4$, that is, $W = [0.5 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 1]^T$. It is observed that after three iteration steps, that is, after 12 s, the critic neural network parameters have effectively converged. Then, the controller is close to the optimal controller $u^*(x) = -(x_2 + 2x_2^3)\sin x_1$. The approximate solution to the HJB equation was determined

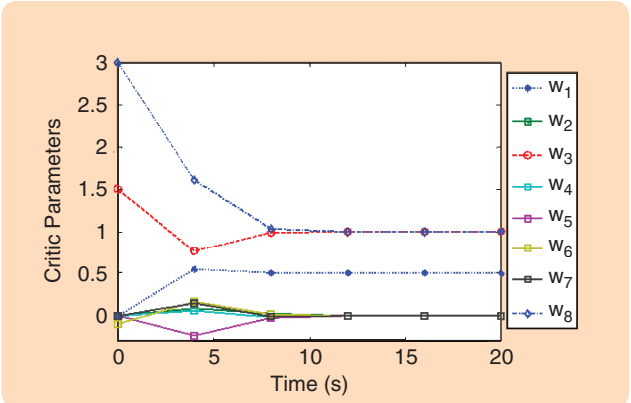

FIGURE 13 Critic neural network parameters, which shows that the IRL adaptive controller converges online to the optimal Riccati equation solution without knowing the system matrix A.

online, and optimal control was found without knowing the system drift dynamics $f(x)$. Note that analytic solution of the HJB equation in this example would be intractable.

SYNCHRONOUS OPTIMAL ADAPTIVE CONTROL FOR CONTINUOUS-TIME SYSTEMS

The aforementioned IRL controller tunes the critic neural network to determine the value while holding the control policy fixed. Then, a policy update is performed. Now an adaptive controller is described that has two neural networks, one for value function approximation and one to approximate the control, which could be called the *critic neural network* and *actor neural network*. The two neural networks are tuned simultaneously, that is, synchronously in time. This procedure is more nearly in line with accepted practice in adaptive control. Though this synchronous controller does require knowledge of the dynamics, it converges to the approximate local solutions to the HJB equation and the Bellman equation online, yet does not require explicitly solving either equation. The HJB is usually impossible to solve for nonlinear systems except for special cases.

Based on the continuous-time Hamiltonian (87) and the stationarity condition $0 = \partial H(x, u, \nabla V^{\mu})/\partial u$, a policy iteration algorithm for continuous-time systems could be written based on the policy evaluation step

$$\begin{aligned} 0 &= H(x, \mu_j(x), \nabla V_{j+1}) \\ &= r(x, \mu_j(x)) + (\nabla V_{j+1})^T (f(x) + g(x)\mu_j(x)), \\ & \qquad V_{j+1}(0) = 0 \end{aligned} \tag{103}$$

and the policy improvement step

$$\mu_{j+1} = \arg\min_{\mu} H(x, \mu, \nabla V_{j+1}). \tag{104}$$

Unfortunately, (103) is a nonlinear partial-differential equation and cannot usually be solved analytically.

However, this policy iteration algorithm provides the structure needed to develop another adaptive control algorithm that can be implemented online using measured data along the trajectories and converges to the optimal control. Specifically, select a value function approximation (VFA), or critic neural network, structure as

$$V(x) = W_1^T \phi(x) \tag{105}$$

and a control action approximation structure or actor neural network as

$$u(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla \phi^T W_2. \tag{106}$$

These approximators could be, for instance, two neural networks with unknown parameters or weights W_1, W_2 and $\phi(x)$ the basis set or activation functions of the first neural network. The structure of the second action neural network comes from (96). Then tuning the neural network weights as

$$\dot{W}_1 = -\alpha_1 \frac{\sigma}{(\sigma^T \sigma + 1)^2} [\sigma^T W_1 + Q(x) + u^T R u], \tag{107}$$

$$\dot{W}_2 = -\alpha_2 \{(F_2 W_2 - F_1 \bar{\sigma}^T W_1) - \frac{1}{4} D(x) W_2 m^T(x) W_1\}, \tag{108}$$

guarantees stability as well as convergence to the optimal value and control [57].

In these parameter estimation algorithms, $\alpha_1, \alpha_2, F_1, F_2$ are algorithm tuning parameters, $D(x) = \nabla \phi(x) g(x) R^{-1} g^T(x) \nabla \phi^T(x)$, $\sigma = \nabla \phi(f + gu)$, $\bar{\sigma} = \sigma/(\sigma^T \sigma + 1)$, and $m(x) = \sigma/(\sigma^T \sigma + 1)^2$. A persistency of excitation condition on $\bar{\sigma}(t)$ is needed to achieve convergence to the optimal value.

This synchronous policy iteration controller is an adaptive control algorithm that requires full knowledge of the system dynamics $f(x), g(x)$, yet converges to the optimal control solution. That is, the algorithm locally approximately solves the HJB equation, which is usually intractable for nonlinear systems. In the continuous-time LQR case, the algorithm solves the ARE using data measured along the trajectories and knowledge of (A, B). The utility of this algorithm is that it can approximately solve the HJB equation for nonlinear systems using data measured along the system trajectories in real time. The HJB is usually impossible to solve for nonlinear systems except in some special cases.

The VFA tuning algorithm for W_1 is based on gradient descent, while the control action tuning algorithm is a form of backpropagation [19] that is, however, also tuned by the VFA weights W_1. This adaptive structure is similar to the actor-critic reinforcement learning structure in Figure 1. However, in contrast to IRL, this algorithm is a continuous-time optimal adaptive controller with two parameter estimators tuned *simultaneously*, that is, synchronously and continuously in time.

Example 5. Continuous-Time Synchronous Optimal Adaptive Control

This example demonstrates the use of the synchronous optimal adaptive control algorithm to approximately solve the HJB equation for a nonlinear continuous-time system by using data measured along the trajectories in real time. This example was developed using [56].

Consider the affine in control input nonlinear system $\dot{x} = f(x) + g(x)u, x \in R^2$, where

$$f(x) = \begin{bmatrix} -x_1 + x_2 \\ -x_1^3 - x_2 - \dfrac{x_1^2}{x_2} + 0.25 x_2 (\cos(2x_1 + x_1^3) + 2)^2 \end{bmatrix}$$

$$g(x) = \begin{bmatrix} 0 \\ \cos(2x_1 + x_1^3) + 2 \end{bmatrix}.$$

and select

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; R = 1.$$

Then the optimal value function that solves the HJB equation is $V^*(x) = 0.25x_1^4 + 0.5x_2^2$ and the optimal control policy is $u^*(x) = -0.5(\cos(2x_1 + x_1^3) + 2)x_2$.

The critic neural network vector activation function are selected as $\phi(x) = [x_1^2 \;\; x_2^2 \;\; x_1^4 \;\; x_2^4]$. The tuning algorithms (107), (108) were run for the critic neural network and control actor neural network respectively, simultaneously in time. A probing noise was added to the control to guarantee persistence of excitation. This noise was decayed exponentially during the simulation. The evolution of the states is shown in Figure 14. The states are bounded and approach zero as the probing noise decays to zero.

Figure 15 shows the critic parameters, denoted by $W_1 = [W_{c1} \;\; W_{c2} \;\; W_{c3} \;\; W_{c4}]^T$ After 80 s, the critic neural network parameters converged to $W_1(t_f) = [0.0033 \;\; 0.4967 \;\; 0.2405 \;\; 0.0153]^T$, which is close to the true weights corresponding to the optimal value $V^*(x)$ that solve the HJB equation. The actor neural network parameters converged to $W_2(t_f) = [0.0033 \;\; 0.4967 \;\; 0.2405 \;\; 0.0153]^T$. The control policy converged to the optimal control

$$\hat{u}_2(x) = -\frac{1}{2}\begin{bmatrix} 0 \\ \cos(2x_1 + x_1^3 + 2) \end{bmatrix}^T \begin{bmatrix} 2x_1 & 0 & 4x_1^3 & 0 \\ 0 & 2x_2 & 0 & 4x_2^3 \end{bmatrix} \hat{W}_2(t_f).$$

Figure 16 shows the three-dimensional (3-D) plot of the difference between the approximated value function by using the online synchronous adaptive algorithm, and the optimal value. This error is small relative to the magnitude of the

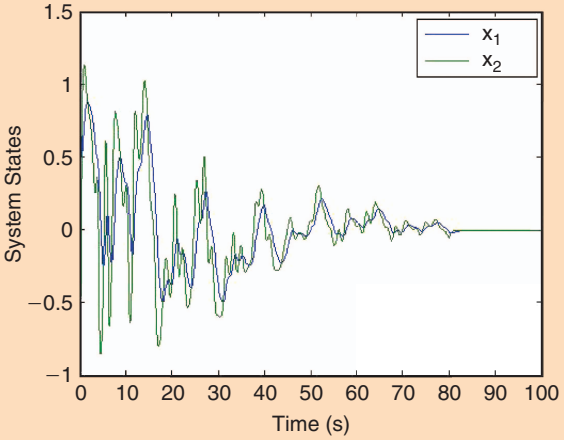

FIGURE 14 Evolution of the states, which shows that the synchronous optimal adaptive controller ensures stability and regulates the states to zero.

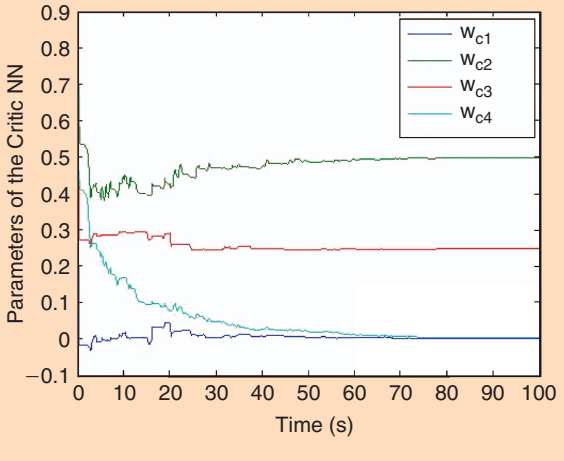

FIGURE 15 Convergence of critic neural network parameters, which shows that the optimal adaptive controller converges to the approximate solution of the nonlinear Hamilton–Jacobi–Bellman equation.

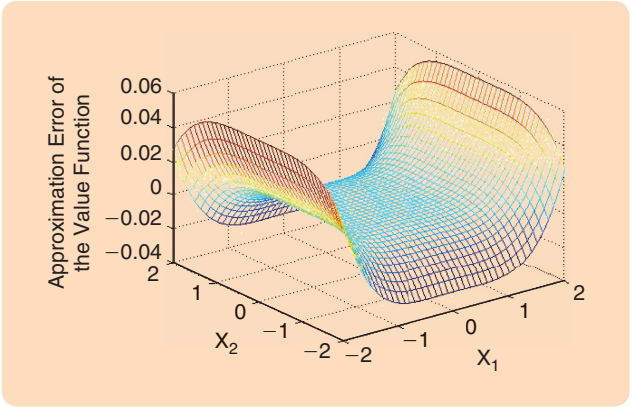

FIGURE 16 Error between optimal and approximated value function. This 3-D plot of the value function error shows that the synchronous optimal adaptive controller converges to a value function that is very close to the true solution of the Hamilton–Jacobi–Bellman equation.

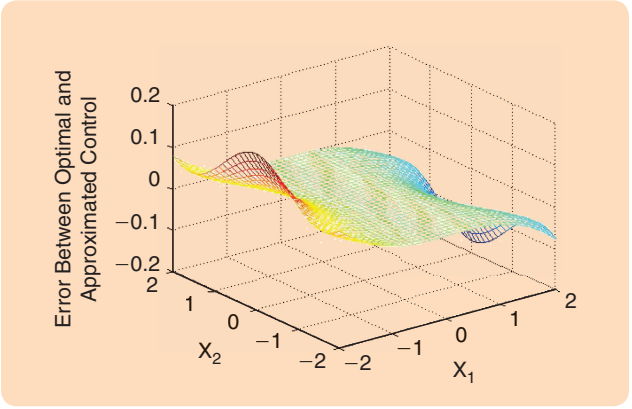

FIGURE 17 Error between optimal and approximated control input. This 3-D plot of the feedback control policy error shows that the synchronous optimal adaptive controller converges very close to the true optimal control policy.

value function. Figure 17 shows the 3-D plot of the difference between the approximated feedback control policy found by using the online algorithm and the optimal control.

This example demonstrates the use of the synchronous optimal adaptive controller to approximately solve an HJB equation online by using data measured along the system trajectories. The HJB equation for this example is intractable to solve analytically.

OPTIMAL ADAPTIVE CONTROL FOR MULTI-PLAYER GAMES AND H-INFINITY CONTROL

The ideas presented in this article can be applied to multiplayer games and H-infinity control. Reinforcement learning techniques have been applied to design adaptive controllers that converge to the solution of two-player zero-sum games in [58], and of multiplayer nonzero-sum games in [59]. In these cases, the adaptive control structure has multiple loops, with action networks and critic networks for each player. The adaptive controller for zero-sum games finds the solution to the H-infinity control problem online in real time. A Q-learning approach to finding the H-infinity controller online is given in [60]. This adaptive controller does not require any systems dynamics information.

CONCLUSION

This article uses computational intelligence techniques to bring together adaptive control and optimal control. Adaptive controllers do not generally converge to optimal solutions, and optimal controllers are designed offline using full dynamics information by solving matrix design equations. The article describes methods from reinforcement learning that can be used to design new types of adaptive controllers that converge to optimal control solutions online in real time by measuring data along the system trajectories. These *optimal adaptive controllers* have multiloop, multitimescale structures that come from the reinforcement learning methods of policy iteration and value iteration. These controllers learn the solutions to Hamilton-Jacobi design equations such as the Riccati equation online without knowing the full dynamical model of the system. A method known as Q learning allows the learning of optimal control solutions online, in the discrete-time case, for completely unknown systems. Q learning has not yet been fully investigated for continuous-time systems.

ACKNOWLEDGMENTS

Support is acknowledged by NSF grant ECCS-1128050, ARO grant W91NF-05-1-0314, and AFOSR grant FA9550-09-1-0278.

AUTHOR INFORMATION

Frank L. Lewis (lewis@uts.edu) received the Ph.D. degree from the Georgia Institute of Technology in 1981. He is the Moncrief-O'Donnell Endowed Chair at the Automation and Robotics Research Institute of The University of Texas at Arlington. He is the coauthor of several books including *Optimal Control*, *Optimal Estimation*, *Aircraft Control and Simulation*, and *Neural Network Control of Robot Manipulators and Nonlinear Systems*. His interests are nonlinear control, optimal control, adaptive control, and intelligent control. He can be contacted at the University of Texas at Arlington Research Institute (UTARI), The University of Texas at Arlington, 7300 Jack Newell Blvd. S, Ft. Worth, Texas 76118-7115 USA.

Draguna Vrabie received the Ph.D. degree from The University of Texas at Arlington and is now a senior research scientist at United Technologies Research Center, East Hartford, Connecticut. Her interests are reinforcement learning, approximate dynamic programming, optimal control, and industrial process control.

Kyriakos G. Vamvoudakis received the Ph.D. degree in electrical engineering from The University of Texas at Arlington in 2011. He is now a project research scientist at the Center of Control, Dynamical Systems, and Computation, at the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. He is coauthor of one patent application, six book chapters, 40 technical publications, and the book *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*. He is the recipient of several awards. His research interests include approximate dynamic programming, game theory, neural network feedback control, and optimal control. Recently, his research has focused on network security and multiagent optimization.

REFERENCES

[1] K. J. Astrom and B. Wittenmark, *Adaptive Control*. Reading, MA: Addison-Wesley, 1995.

[2] P. Ioannou and B. Fidan, *Adaptive Control Tutorial*. Philadelphia, PA: SIAM Press, 2006.

[3] F. L. Lewis, D. Vrabie, and V. Syrmos, *Optimal Control*, 3rd ed. New York: Wiley, 2012.

[4] Z.-H. Li and M. Krstic, "Optimal design of adaptive tracking controllers for nonlinear systems," *Automatica*, vol. 33, no. 8, pp. 1459–1473, 1997.

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

[6] W. B. Powell, *Approximate Dynamic Programming*. Hoboken, NJ: Wiley, 2007.

[7] P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1991, pp. 67–95.

[8] F. L. Lewis, G. Lendaris, and D. Liu, "Special issue on adaptive dynamic programming and reinforcement learning for feedback control," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 896–897, Aug. 2008.

[9] X. Cao, *Stochastic Learning and Optimization*. Berlin, Germany: Springer-Verlag, 2007.

[10] J. M. Mendel and R. W. MacLaren, "Reinforcement learning control and pattern recognition systems," in *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications*, J. M. Mendel and K. S. Fu, Eds. New York: Academic, 1970, pp. 287–318.

[11] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2009.

[12] W. Schultz, "Neural coding of basic reward terms of animal learning theory, game theory, microeconomics and behavioral ecology," *Current Opinion Neurobiol.*, vol. 14, no. 2, pp. 139–147, 2004.

[13] K. Doya, H. Kimura, and M. Kawato, "Neural mechanisms for learning and control," *IEEE Control Syst. Mag.*, vol. 21, no. 4, pp. 42–54, Aug. 2000.

[14] P. J. Werbos, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992.

[15] D. Vrabie and F. L. Lewis, "Neural network approach to continuous-time direct adaptive optimal control for partially-unknown nonlinear systems," *Neural Netw.*, vol. 22, no. 3, pp. 237–246, Apr. 2009.

[16] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834–846, Sep./Oct. 1983.

[17] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[18] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[19] P. J. Werbos, "Neural networks for control and system identification," in *Proc. IEEE Conf. Decision Control*, Tampa, FL, 1989, pp. 260–265.

[20] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997.

[21] J. Si, A. Barto, W. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming*. Piscataway, NJ: IEEE Press, 2004.

[22] S. N. Balakrishnan, J. Ding, and F. L. Lewis, "Issues on stability of ADP feedback controllers for dynamical systems," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 913–917, Aug. 2008.

[23] F. Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Comput, Intell, Mag.*, vol. 4, no. 2, pp. 39–47, May 2009.

[24] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32–50, 2009.

[25] D. Han and S. N. Balakrishnan, "State-constrained agile missile control with adaptive-critic-based neural networks," *IEEE Trans. Control Syst. Technol.*, vol. 10, no. 4, pp. 481–489, Jul. 2002.

[26] D. Prokhorov, *Computational Intelligence in Automotive Applications*. New York: Springer-Verlag, 2008.

[27] S. Ferrari and R. F. Stengel, "An adaptive critic global controller," in *Proc. American Control Conf.*, May 2002, pp. 2665–2670.

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

[29] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Trans. Syst., Man Cybern. C*, vol. 32, no. 2, pp. 140–153, 2002.

[30] R. Enns and J. Si, "Helicopter flight control reconfiguration for main rotor actuator failures," *AIAA J. Guidance, Control, Dynamics*, vol. 26, no. 4, pp. 572–584, 2003.

[31] C. Lu, J. Si, and X. Xie, "Direct heuristic dynamic programming method for power system stability enhancement," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 1008–1013, 2008.

[32] G. G. Lendaris, L. Schultz, and T. Shannon, "Adaptive critic design for intelligent steering and speed control of a 2-axle vehicle," in *Proc. Int. Conf. Neural Networks*, 2000, pp. 73–78.

[33] X. Liu and S. N. Balakrishnan, "Convergence analysis of adaptive critic based optimal control," in *Proc. American Control Conf.*, June 2000, pp. 1929–1933.

[34] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 943–949, Aug. 2008.

[35] C. Darwin, *On the Origin of Species by Means of Natural Selection*. London, U.K.: John Murray, 1859.

[36] D. G. Luenberger, *Introduction to Dynamic Systems*. New York: Wiley, 1979.

[37] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. L. Lewis, "Adaptive optimal control for continuous-time linear systems based on policy iteration," *Automatica*, vol. 45, no. 2, pp. 477–484, 2009.

[38] A. Papoulis, *Probability Random Variables and Stochastic Processes*. New York: McGraw-Hill, 2002.

[39] R. M. Wheeler and K. S. Narendra, "Decentralized learning in finite Markov chains," *IEEE Trans. Autom. Control*, vol. 31, no. 6, pp. 519–526, June 1986.

[40] P. Mehta and S. Meyn, "Q-learning and Pontryagin's minimum principle," in *Proc. IEEE Conf. Decision Control*, Dec. 2009, pp. 3598–3605.

[41] H. Zhang, J. Huang, and F. L. Lewis, "Algorithm and stability of ATC receding horizon control," in *Proc. IEEE Symp. Adaptive Dynamic Programming Reinforcement*, Nashville, TN, Mar. 2009, pp. 28–35.

[42] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, Cambridge, U.K., 1989.

[43] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[44] G. A. Hewer, "An iterative technique for the computation of the steady state gains for the discrete optimal regulator," *IEEE Trans Autom. Control*, vol. 16, no. 4, pp. 382–384, Aug. 1971.

[45] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*. London, U.K.: Oxford Univ. Press, 1995.

[46] K. L. Moore, *Iterative Learning Control for Deterministic Systems*. London, U.K.: Springer-Verlag, 1993.

[47] A. J. Van, "L2-gain analysis of nonlinear systems and nonlinear state feedback H∞ control," *IEEE Trans. Autom. Control*, vol. 37, no. 6, pp. 770–784, 1992.

[48] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[49] S. Bradtke, B. Ydstie, and A. Barto, "Adaptive linear quadratic control using policy iteration," in *Proc. Amer. Control Conf.*, Baltimore, MD, 1994, pp. 3475–3479.

[50] F. L. Lewis and K. G. Vamvoudakis, "Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using measured output data," *IEEE Trans. Syst., Man, Cybern. B*, vol. 41, no. 1, pp. 14–25, Feb. 2011.

[51] M. Abu-Khalaf, F. L. Lewis, and J. Huang, "Policy iterations on the Hamilton-Jacobi-Isaacs equation for state feedback control with input saturation," *IEEE Trans. Autom. Control*, vol. 51, no. 12, pp. 1989–1995, Dec. 2006.

[52] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Proc. Int. Conf. Neural Networks*, Orlando, FL, June1994, pp. 2448–2453.

[53] K. Doya, "Reinforcement learning in continuous time and space," *Neural Comput.*, vol. 12, no. 1, pp. 219–245, 2000.

[54] T. Hanselmann, L. Noakes, and A. Zaknich, "Continuous-time adaptive critics," *IEEE Trans. Neural Netw.*, vol. 18, no. 3, pp. 631–647, May 2007.

[55] D. L. Kleinman, "On an iterative technique for Riccati equation computations," *IEEE Trans. Autom. Control*, vol. AC–13, no. 1, pp. 114–115, Feb. 1968.

[56] V. Nevistic and J. Primbs, "Constrained nonlinear optimal control: A converse HJB approach," Dept. Control Dynamical Systems, California Institute of Technology, Pasadena, CA, Tech. Rep. 96–021, 1996.

[57] K. G. Vamvoudakis and F. L. Lewis, "Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem," *Automatica*, vol. 46, no. 5, pp. 878–888, 2010.

[58] D. Vrabie and F. L. Lewis, "Adaptive dynamic programming for online solution of a zero-sum differential game," *J. Control Theory: Its Appl.*, vol. 9, no. 3, pp. 353–360, 2011.

[59] K. G. Vamvoudakis and F. Lewis, "Multi-player non-zero sum games: Online adaptive learning solution of coupled Hamilton-Jacobi equations," *Automatica*, vol. 47, no. 8, pp. 556–569, 2011.

[60] J. H. Kim and F. L. Lewis, "Model-free H-infinity control design for unknown linear discrete-time systems via Q-learning with LMI," *Automatica*, vol. 46, no. 8, pp. 1320–1326, Aug. 2010.