A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients

Ivo Grondman, Lucian Buşoniu, Gabriel A. D. Lopes, and Robert Babuška

***Abstract*—Policy-gradient-based actor-critic algorithms are amongst the most popular algorithms in the reinforcement learning framework. Their advantage of being able to search for optimal policies using low-variance gradient estimates has made them useful in several real-life applications, such as robotics, power control, and finance. Although general surveys on reinforcement learning techniques already exist, no survey is specifically dedicated to actor-critic algorithms in particular. This paper, therefore, describes the state of the art of actor-critic algorithms, with a focus on methods that can work in an online setting and use function approximation in order to deal with continuous state and action spaces. After starting with a discussion on the concepts of reinforcement learning and the origins of actor-critic algorithms, this paper describes the workings of the natural gradient, which has made its way into many actor-critic algorithms over the past few years. A review of several standard and natural actor-critic algorithms is given, and the paper concludes with an overview of application areas and a discussion on open issues.**

***Index Terms*—Actor-critic, natural gradient, policy gradient, reinforcement learning (RL).**

I. Introduction

REINFORCEMENT learning (RL) is a framework in which an agent (or controller) optimizes its behavior by interacting with its environment. After taking an action in some state, the agent receives a scalar reward from the environment, which gives the agent an indication of the quality of that action. The function that indicates the action to take in a certain state is called the *policy*. The main goal of the agent is to find a policy that maximizes the total accumulated reward, also called the *return*. By following a given policy and processing the rewards, the agent can build estimates of the return. The function representing this estimated return is known as the *value function*. Using this value function allows the agent to make indirect use of past experiences to decide on future actions to take in or around a certain state.

Over the course of time, several types of RL algorithms have been introduced, and they can be divided into three groups [1]: actor-only, critic-only, and actor-critic methods, where the words actor and critic are synonyms for the policy and value function, respectively. Actor-only methods typically work with a parameterized family of policies over which optimization procedures can be used directly. The benefit of a parameterized policy is that a spectrum of continuous actions can be generated, but the optimization methods used (typically policy gradient methods) suffer from high variance in the estimates of the gradient, leading to slow learning [1]–[5].

Critic-only methods that use temporal difference (TD) learning have a lower variance in the estimates of expected returns [3], [5], [6]. A straightforward way of deriving a policy in critic-only methods is by selecting *greedy actions* [7]: actions for which the value function indicates that the expected return is the highest. However, to do this, one needs to resort to an optimization procedure in every state encountered to find the action leading to an optimal value. This can be computationally intensive, especially if the action space is continuous. Therefore, critic-only methods usually discretize the continuous action space, after which the optimization over the action space becomes a matter of enumeration. Obviously, this approach undermines the ability of using continuous actions and thus of finding the true optimum.

Actor-critic methods combine the advantages of actor-only and critic-only methods. While the parameterized actor brings the advantage of computing continuous actions without the need for optimization procedures on a value function, the critic's merit is that it supplies the actor with low-variance knowledge of the performance. More specifically, the critic's estimate of the expected return allows for the actor to update with gradients that have lower variance, speeding up the learning process. The lower variance is traded for a larger bias at the start of learning when the critic's estimates are far from accurate [5]. Actor-critic methods usually have good convergence properties, in contrast with critic-only methods [1].

These nice properties of actor-critic methods have made them a preferred RL algorithm, also in real-life application domains. General surveys on RL already exist [8]–[10], but because of the growing popularity and recent developments in the field of actor-critic algorithms, this class of reinforcement algorithms deserves a survey in its own right. The goal of this paper is to give an overview of the work on (online) actor-critic algorithms, giving technical details of some representative algorithms, as well as to provide references to a number of application papers. Additionally, the algorithms are presented in one unified notation, which allows for a better technical comparison of the variants and implementations. Because the discrete-time variant has been developed to a reasonable level of maturity, this paper solely discusses algorithms in the discrete-time setting. Continuous-time

Manuscript received August 8, 2011; revised June 28, 2012; March 20, 2012; accepted August 8, 2012. Date of current version December 17, 2012. This paper was recommended by Associate Editor G. I. Papadimitriou.

I. Grondman, G. A. D. Lopes, and R. Babuška are with the Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: i.grondman@tudelft.nl; g.a.delgadolopes; r.babuska@tudelft.nl).

L. Buşoniu is with Université de Lorraine, CRAN, UMR 7039 and CNRS, CRAN, UMR 7039, 54500 Vandoeuvre-lès-Nancy, France. He is also associated with the Department of Automation, Technical University of Cluj-Napoca, 400020 Cluj-Napoca, Romania (e-mail: lucian@busoniu.net).

Digital Object Identifier 10.1109/TSMCC.2012.2218595

variants of actor-critic algorithms (see, e.g., [11] and [12]) and multiagent actor-critic schemes [13], [14] are not considered here.

The focus is put on actor-critic algorithms based on policy gradients, which constitute the largest part of actor-critic algorithms. A distinction is made between algorithms that use a *standard* (sometimes also called vanilla) gradient and the *natural* gradient that became more popular in the course of the last decade. The remaining part of actor-critic algorithms consists mainly of algorithms that update a policy by moving it toward the greedy policy underlying an approximate state-action value function [10]. In this paper, these algorithms are regarded as critic-only algorithms as the policy is implemented implicitly by the critic. Algorithms are only categorized as actor-critic here if they implement two separately parameterized representations for the actor and the critic. Furthermore, all algorithms make use of function approximation, which in real-life applications such as robotics is necessary in order to deal with continuous state and action spaces.

This paper is organized as follows. Section II introduces the basic concepts of a Markov decision process (MDP), which is the cornerstone of RL. Section III describes critic-only, actor-only, and actor-critic RL algorithms and the important policy gradient theorem, after which Section IV surveys actor-critic algorithms that use a standard gradient. Section V describes the natural gradient and its application to actor-critic methods, as well as surveys several natural actor-critic algorithms. Section VI briefly reviews the application areas of these methods. A discussion and future outlook is provided in Section VII.

II. Markov Decision Processes

This section introduces the concepts of discrete-time RL, based on [7], but extended to the use of continuous state and action spaces and also assuming a stochastic setting, as covered more extensively in [15] and [16].

An RL algorithm can be used to solve problems modeled as MDPs. An MDP is a tuple $\langle X, U, f, \rho \rangle$, where X denotes the state space, U the action space, $f : X \times U \times X \mapsto [0, \infty)$ the state transition probability density function, and $\rho : X \times U \times X \mapsto \mathbb{R}$ the reward function. In this paper, only stationary MDPs are considered, i.e., the elements of the tuple $\langle X, U, f, \rho \rangle$ do not change over time.

The stochastic process to be controlled is described by the state transition probability density function f. It is important to note that since state space is continuous, it is only possible to define a probability of reaching a certain state *region*, since the probability of reaching a particular state is zero. The probability of reaching a state x_{k+1} in the region $X_{k+1} \subseteq X$ from state x_k after applying action u_k is

$$P(x_{k+1} \in X_{k+1} | x_k, u_k) = \int_{X_{k+1}} f(x_k, u_k, x')dx'.$$

After each transition to a state x_{k+1}, the controller receives an immediate reward

$$r_{k+1} = \rho(x_k, u_k, x_{k+1})$$

which depends on the previous state, the current state, and the action taken. The reward function ρ is assumed to be bounded. The action u_k taken in a state x_k is drawn from a stochastic policy $\pi : X \times U \mapsto [0, \infty)$.

The goal of the RL agent is to find the policy π which maximizes the expected value of a certain function g of the immediate rewards received while following the policy π. This expected value is the cost-to-go function

$$J(\pi) = \mathrm{E}\left\{ g(r_1, r_2, \ldots) | \pi \right\}.$$

In most cases,[1] the function g is either the discounted sum of rewards or the average reward received, as explained next.

A. Discounted Reward

In the discounted reward setting [18], the cost function J is equal to the expected value of the discounted sum of rewards when starting from an initial state $x_0 \in X$ drawn from an initial state distribution $x_0 \sim d_0(\cdot)$; this sum is also called the discounted return

$$J(\pi) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| d_0, \pi \right\}$$
$$= \int_X d_\gamma^\pi(x) \int_U \pi(x,u) \int_X f(x,u,x')\rho(x,u,x')dx'dudx, \quad (1)$$

where $d_\gamma^\pi(x) = \sum_{k=0}^{\infty} \gamma^k p(x_k = x | d_0, \pi)$ is the discounted state distribution under the policy π [16], [19], and $\gamma \in [0, 1)$ denotes the reward discount factor. Note that $p(x_k = x)$ is a probability density function here.

During learning, the agent will have to estimate the cost-to-go function J for a given policy π. This procedure is called *policy evaluation*. The resulting estimate of J is called the *value function* and two definitions exist for it. The state value function

$$V^\pi(x) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| x_0 = x, \pi \right\} \quad (2)$$

only depends on the state x and assumes that the policy π is followed starting from this state. The state-action value function

$$Q^\pi(x,u) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| x_0 = x, u_0 = u, \pi \right\} \quad (3)$$

also depends on the state x, but makes the action u chosen in this state a free variable instead of having it generated by the policy π. Once the first transition onto a next state has been made, π governs the rest of the action selection. The relationship between these two definitions for the value function is given by

$$V^\pi(x) = E\left\{ Q^\pi(x,u) | \, u \sim \pi(x, \cdot) \right\}.$$

With some manipulation, (2) and (3) can be put into a recursive form [18]. For the state value function, this is

$$V^\pi(x) = E\left\{ \rho(x,u,x') + \gamma V^\pi(x') \right\} \quad (4)$$

[1]Other cost functionals do exist and can be used for actor-critic algorithms, such as the risk-sensitive cost in [17].

with u drawn from the probability distribution function $\pi(x, \cdot)$ and x' drawn from $f(x, u, \cdot)$. For the state-action value function, the recursive form is

$$Q^{\pi}(x,u) = E\left\{\rho(x,u,x') + \gamma Q^{\pi}(x',u')\right\} \tag{5}$$

with x' drawn from the probability distribution function $f(x, u, \cdot)$ and u' drawn from the distribution $\pi(x', \cdot)$. These recursive relationships are called Bellman equations [7].

Optimality for both the state value function V^{π} and the state-action value function Q^{π} is governed by the Bellman *optimality* equation. Denoting the optimal state value function with $V^{*}(x)$ and the optimal state-action value with $Q^{*}(x,u)$, the corresponding Bellman optimality equations for the discounted reward setting are

$$V^{*}(x) = \max_{u} E\left\{\rho(x,u,x') + \gamma V^{*}(x')\right\} \tag{6a}$$

$$Q^{*}(x,u) = E\left\{\rho(x,u,x') + \gamma \max_{u'} Q^{*}(x',u')\right\}. \tag{6b}$$

B. Average Reward

As an alternative to the discounted reward setting, there is also the approach of using the *average* return [18]. In this setting, a distribution d_0 does not need to be chosen, under the assumption that the process is ergodic [7] and thus that J does not depend on the starting state. Instead, the value functions for a policy π are defined relative to the average expected reward per time step under the policy, turning the cost-to-go function into

$$\begin{aligned} J(\pi) &= \lim_{n\to\infty} \frac{1}{n} E\left\{ \sum_{k=0}^{n-1} r_{k+1} \,\middle|\, \pi \right\} \\ &= \int_X d^{\pi}(x) \int_U \pi(x,u) \int_X f(x,u,x')\rho(x,u,x') dx' du dx. \end{aligned} \tag{7}$$

Equation (7) is very similar to (1), except that the definition for the state distribution changed to $d^{\pi}(x) = \lim_{k\to\infty} p(x_k = x, \pi)$. For a given policy π, the state value function $V^{\pi}(x)$ and state-action value function $Q^{\pi}(x,u)$ are then defined as

$$V^{\pi}(x) = E\left\{ \sum_{k=0}^{\infty} (r_{k+1} - J(\pi)) \,\middle|\, x_0 = x, \pi \right\}$$

$$Q^{\pi}(x,u) = E\left\{ \sum_{k=0}^{\infty} (r_{k+1} - J(\pi)) \,\middle|\, x_0 = x, u_0 = u, \pi \right\}.$$

The Bellman equations for the average reward—in this case also called the Poisson equations [20]—are

$$V^{\pi}(x) + J(\pi) = E\left\{\rho(x,u,x') + V^{\pi}(x')\right\} \tag{8}$$

with u and x' drawn from the appropriate distributions as before and

$$Q^{\pi}(x,u) + J(\pi) = E\left\{\rho(x,u,x') + Q^{\pi}(x',u')\right\} \tag{9}$$

again with x' and u' drawn from the appropriate distributions. Note that (8) and (9) both require the value $J(\pi)$, which is unknown and hence needs to be estimated in some way. The Bellman optimality equations, describing an optimum for the average reward case, are

$$V^{*}(x) + J^{*} = \max_{u} E\left\{\rho(x,u,x') + V^{*}(x')\right\} \tag{10a}$$

$$Q^{*}(x,u) + J^{*} = E\left\{\rho(x,u,x') + \max_{u'} Q^{*}(x',u')\right\} \tag{10b}$$

where J^{*} is the optimal average reward as defined by (7) when an optimal policy π^{*} is used.

III. Actor-Critic in the Context of Reinforcement Learning

As discussed in the introduction, the vast majority of RL methods can be divided into three groups [1]: critic-only, actor-only, and actor-critic methods. This section will give an explanation on all three groups, starting with critic-only methods. The part on actor-only methods introduces the concept of a policy gradient, which provides the basis for actor-critic algorithms. The final part of this section explains the policy gradient theorem, an important result that is now widely used in many implementations of actor-critic algorithms.

In real-life applications, such as robotics, processes usually have continuous state and action spaces, making it impossible to store exact value functions or policies for each separate state or state-action pair. Any RL algorithm used in practice will have to make use of function approximators for the value function and/or the policy in order to cover the full range of states and actions. Therefore, this section assumes the use of such function approximators.

A. Critic-Only Methods

Critic-only methods, such as Q-learning [21]–[23] and SARSA [24], use a state-action value function and no explicit function for the policy. For continuous state and action spaces, this will be an approximate state-action value function. These methods learn the optimal value function by finding online an approximate solution to the Bellman equations (6b) or (10b). A deterministic policy, denoted by $\pi : X \mapsto U$, is calculated by using an optimization procedure over the value function

$$\pi(x) = \arg\max_{u} Q(x,u). \tag{11}$$

There is no reliable guarantee on the near-optimality of the resulting policy for just any approximated value function when learning in an online setting. For example, Q-learning and SARSA with specific function approximators have been shown not to converge even for simple MDPs [25]–[27]. However, the counterexamples used to show divergence were further analyzed in [28] (with an extension to the stochastic setting in [29]), and it was shown that convergence can be assured for linear-in-parameters function approximators if trajectories are sampled according to their on-policy distribution. The work in [28] also provides a bound on the approximation error between the true value function and the approximation learned by online TD learning. An analysis of more approximate policy evaluation methods is provided in [30], mentioning conditions for convergence and bounds on the approximation error for each method. Nevertheless, for most choices of basis functions, an approximated value function learned by TD learning will be biased. This is reflected by the state-of-the-art bounds on the least-squares temporal difference (LSTD) solution quality [31], which always include a term depending on the distance between the true value function and its projection on the approximation space. For a particularly bad choice of basis functions, this bias can grow very large.

B. Actor-Only Methods and the Policy Gradient

Policy gradient methods (see, for instance, the SRV [32] and Williams' REINFORCE algorithms [33]) are principally actor-only and do not use any form of a stored value function. Instead, the majority of actor-only algorithms work with a parameterized family of policies and optimize the cost defined by (1) or (7) directly over the parameter space of the policy. Although not explicitly considered here, work on nonparametric policy gradients does exist (see, e.g., [34] and [35]). A major advantage of actor-only methods over critic-only methods is that they allow the policy to generate actions in the complete continuous action space.

A policy gradient method is generally obtained by parameterizing the policy π by the parameter vector $\vartheta \in \mathbb{R}^p$. Considering that both (1) and (7) are functions of the parameterized policy π_ϑ, they are in fact functions of ϑ. Assuming that the parameterization is differentiable with respect to ϑ, the gradient of the cost function with respect to ϑ is described by

$$\nabla_\vartheta J = \frac{\partial J}{\partial \pi_\vartheta} \frac{\partial \pi_\vartheta}{\partial \vartheta}. \tag{12}$$

Then, by using standard optimization techniques, a locally optimal solution of the cost J can be found. The gradient $\nabla_\vartheta J$ is estimated per time step, and the parameters are then updated in the direction of this gradient. For example, a simple gradient ascent method would yield the policy gradient update equation

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k} \nabla_\vartheta J_k \tag{13}$$

where $\alpha_{a,k} > 0$ is a small enough learning rate for the actor, by which it is obtained that[2] $J(\vartheta_{k+1}) \geq J(\vartheta_k)$.

Several methods exist to estimate the gradient, e.g., by using infinitesimal perturbation analysis or likelihood-ratio methods [36], [37]. For a broader discussion on these methods, see [4] and [38]. Approaches to model-based gradient methods are given in [39]–[41] and in the more recent work of Deisenroth and Rasmussen [42].

The main advantage of actor-only methods is their strong convergence property, which is naturally inherited from gradient descent methods. Convergence is obtained if the estimated gradients are unbiased and the learning rates $\alpha_{a,k}$ satisfy [7], [38]

$$\sum_{k=0}^{\infty} \alpha_{a,k} = \infty, \qquad \sum_{k=0}^{\infty} \alpha_{a,k}^2 < \infty.$$

[2]One could also define the cost J such that it should be minimized. In that case, the plus sign in (13) is replaced with a minus sign, resulting in $J(\vartheta_{k+1}) \leq J(\vartheta_k)$.

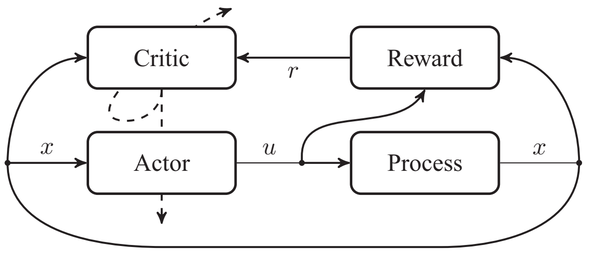

Fig. 1. Schematic overview of an actor-critic algorithm. The dashed line indicates that the critic is responsible for updating the actor and itself.

A drawback of the actor-only approach is that the estimated gradient may have a large variance [19], [43]. In addition, every gradient is calculated without using any knowledge of past estimates [1], [44].

C. Actor-Critic Algorithms

Actor-critic methods [45], [46] aim to combine the advantages of actor-only and critic-only methods. Like actor-only methods, actor-critic methods are capable of producing continuous actions, while the large variance in the policy gradients of actor-only methods is countered by adding a critic. The role of the critic is to evaluate the current policy prescribed by the actor. In principle, this evaluation can be done by any policy evaluation method commonly used, such as TD [6], [18], LSTD [3], [18], [47], or residual gradients [25]. The critic approximates and updates the value function using samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement. These methods usually preserve the desirable convergence properties of policy gradient methods, in contrast with critic-only methods. In actor-critic methods, the policy is not directly inferred from the value function by using (11). Instead, the policy is updated in the policy gradient direction using only a small step size α_a, meaning that a change in the value function will only result in a small change in the policy, leading to less or no oscillatory behavior in the policy, as described in [48].

Fig. 1 shows the schematic structure of an actor-critic algorithm. The learning agent has been split into two separate entities: the actor (policy) and the critic (value function). The actor is only responsible for generating a control input u, given the current state x. The critic is responsible for processing the rewards it receives, i.e., evaluating the quality of the current policy by adapting the value function estimate. After a number of policy evaluation steps by the critic, the actor is updated by using information from the critic.

A unified notation for the actor-critic algorithms described in this paper allows for an easier comparison between them. In addition, most algorithms can be fitted to a general *template* of standard update rules. Therefore, two actor-critic algorithm templates are introduced: one for the discounted reward setting and one for the average reward setting. Once these templates are established, specific actor-critic algorithms can be discussed by only looking at how they fit into the general template or in what way they differ from it.

For both reward settings, the value function is parameterized by the parameter vector $\theta \in \mathbb{R}^q$. This will be denoted with $V_\theta(x)$ or $Q_\theta(x,u)$. If the parameterization is linear, the features (basis functions) will be denoted with ϕ, i.e.,

$$V_\theta(x) = \theta^\top \phi(x) \text{ or } Q_\theta(x,u) = \theta^\top \phi(x,u). \tag{14}$$

The stochastic policy π is parameterized by $\vartheta \in \mathbb{R}^p$ and will be denoted with $\pi_\vartheta(x,u)$. If the policy is denoted with $\pi_\vartheta(x)$, it is deterministic and no longer represents a probability density function, but the direct mapping from states to actions $u = \pi_\vartheta(x)$.

The goal in actor-critic algorithms—or any other RL algorithm for that matter—is to find the best policy possible, given some stationary MDP. A prerequisite for this is that the critic is able to accurately evaluate a given policy. In other words, the goal of the critic is to find an approximate solution to the Bellman equation for that policy. The difference between the right-hand and left-hand sides of the Bellman equation, whether it is the one for the discounted reward setting (4) or the average reward setting (8), is called the TD error and is used to update the critic. Using the function approximation for the critic and a transition sample $(x_k, u_k, r_{k+1}, x_{k+1})$, the TD error is estimated as

$$\delta_k = r_{k+1} + \gamma V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k). \tag{15}$$

Perhaps the most standard way of updating the critic is to exploit this TD error for use in a gradient descent update [7]

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k \nabla_\theta V_{\theta_k}(x_k) \tag{16}$$

where $\alpha_{c,k} > 0$ is the learning rate of the critic. For the linearly parameterized function approximator (14), this reduces to

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k \phi(x_k). \tag{17}$$

This TD method is also known as TD(0) learning, as no eligibility traces are used. The extension to the use of eligibility traces, resulting in TD(λ) methods, is straightforward and is explained next.

Using (16) to update the critic results in a one-step backup, whereas the reward received is often the result of a series of steps. Eligibility traces offer a better way of assigning credit to states or state-action pairs visited several steps earlier. The eligibility trace vector for all q features at time instant k is denoted with $z_k \in \mathbb{R}^q$ and its update equation is [1], [7]

$$z_k = \lambda\gamma z_{k-1} + \nabla_\theta V_{\theta_k}(x_k).$$

It decays with time by a factor $\lambda\gamma$, with $\lambda \in [0,1)$ the trace decay rate. This makes the recently used features more eligible for receiving credit. The use of eligibility traces speeds up the learning considerably. Using the eligibility trace vector z_k, the update (16) of the critic becomes

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k z_k. \tag{18}$$

With the use of eligibility traces, the actor-critic template for the discounted return setting becomes

$$\delta_k = r_{k+1} + \gamma V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k) \tag{19a}$$

$$z_k = \lambda\gamma z_{k-1} + \nabla_\theta V_{\theta_k}(x_k) \tag{19b}$$

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k z_k \tag{19c}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k}\nabla_\vartheta J_k. \tag{19d}$$

Although not commonly seen, eligibility traces may be introduced for the actor as well. As with actor-only methods, several ways exist to estimate $\nabla_\vartheta J_k$.

For the average reward case, the critic can be updated using the average-cost TD method [49]. Then, Bellman equation (8) is considered, turning the TD error into

$$\delta_k = r_{k+1} - \hat{J}_k + V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k)$$

with $\hat{J}_k$ an estimate of the average cost at time k. Obviously, this requires an update equation for the estimate $\hat{J}$ as well, which usually is [1]

$$\hat{J}_k = \hat{J}_{k-1} + \alpha_{J,k}(r_{k+1} - \hat{J}_{k-1})$$

where $\alpha_{J,k} \in (0,1]$ is another learning rate. The critic still updates with (18). The update of the eligibility trace also needs to be adjusted, as the discount rate γ is no longer present. The template for actor-critic algorithms in the average return setting then is

$$\hat{J}_k = \hat{J}_{k-1} + \alpha_{J,k}(r_{k+1} - \hat{J}_{k-1}) \tag{20a}$$

$$\delta_k = r_{k+1} - \hat{J}_k + V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k) \tag{20b}$$

$$z_k = \lambda z_{k-1} + \nabla_\theta V_{\theta_k}(x_k) \tag{20c}$$

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k z_k \tag{20d}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k}\nabla_\vartheta J_k. \tag{20e}$$

For the actor-critic algorithm to converge, it is necessary that the critic's estimate is at least asymptotically accurate. This is the case if the step sizes $\alpha_{a,k}$ and $\alpha_{c,k}$ are deterministic, nonincreasing, and satisfy [1]

$$\sum_k \alpha_{a,k} = \infty, \qquad \sum_k \alpha_{c,k} = \infty \tag{21}$$

$$\sum_k \alpha_{a,k}^2 < \infty, \qquad \sum_k \alpha_{c,k}^2 < \infty, \qquad \sum_k \left(\frac{\alpha_{a,k}}{\alpha_{c,k}}\right)^d < \infty \tag{22}$$

for some $d \geq 0$. The learning rate $\alpha_{J,k}$ is usually set equal to $\alpha_{c,k}$. Note that such assumptions on learning rates are typical for all RL algorithms. They ensure that learning will slow down but never stops and also that the update of the actor operates on a slower time-scale than the critic, to ensure that the critic has enough time to evaluate the current policy.

Although TD(λ) learning is used quite commonly, other ways of determining the critic parameter θ do exist and some are even known to be superior in terms of convergence rate in both discounted and average reward settings [50], such as LSTD [3], [47]. LSTD uses samples collected along a trajectory generated by a policy π to set up a system of TD equations derived from or similar to (19a) or (20b). As LSTD requires an approximation of the value function which is linear in its parameters, i.e.,

$V_\theta(x) = \theta^\top \phi(x)$, this system is linear and can easily be solved for θ by a least-squares method. Regardless of how the critic approximates the value function, the actor update is always centered around (13), using some way to estimate $\nabla_\vartheta J_k$.

For actor-critic algorithms, the question arises how the critic influences the gradient update of the actor. This is explained in the next section about the policy gradient theorem.

D. Policy Gradient Theorem

Many actor-critic algorithms now rely on the policy gradient theorem, a result obtained simultaneously in [1] and [19], proving that an unbiased estimate of the gradient (12) can be obtained from experience using an approximate value function satisfying certain properties. The basic idea, given by Konda and Tsitsiklis [1], is that since the number of parameters that the actor has to update is relatively small compared with the (usually infinite) number of states, it is not useful to have the critic attempting to compute the exact value function, which is also a high-dimensional object. Instead, it should compute a projection of the value function onto a low-dimensional subspace spanned by a set of basis functions, which are completely determined by the parameterization of the actor.

In the case of an approximated stochastic policy, but exact state-action value function Q^π, the policy gradient theorem is as follows.

Theorem 1 (Policy Gradient): For any MDP, in either the average reward or discounted reward setting, the policy gradient is given by

$$\nabla_\vartheta J = \int_X d^\pi(x) \int_U \nabla_\vartheta \pi(x,u) Q^\pi(x,u) du dx$$

with $d^\pi(x)$ defined for the appropriate reward setting.

Proof: See [19]. ■

This clearly shows the relationship between the policy gradient $\nabla_\vartheta J$ and the critic function $Q^\pi(x,u)$ and ties together the update equations of the actor and critic in the templates (19) and (20).

For most applications, the state-action space is continuous and thus infinite, which means that it is necessary to approximate the state(-action) value function. The result in [1] and [19] shows that $Q^\pi(x,u)$ can be approximated with[3] $h_w : X \times U \mapsto \mathbb{R}$, parameterized by w, without affecting the unbiasedness of the policy gradient estimate.

In order to find the closest approximation of Q^π by h_w, one can try to find the w that minimizes the quadratic error

$$\mathcal{E}_w^\pi(x,u) = \frac{1}{2}\left[Q^\pi(x,u) - h_w(x,u)\right]^2.$$

The gradient of this quadratic error with respect to w is

$$\nabla_w \mathcal{E}_w^\pi(x,u) = \left[Q^\pi(x,u) - h_w(x,u)\right] \nabla_w h_w(x,u) \quad (23)$$

and this can be used in a gradient descent algorithm to find the optimal w. If the estimator of $Q^\pi(x,u)$ is unbiased, the expected

[3]This approximation of $Q^\pi(x,u)$ is not denoted with an accented Q as it is not actually the value function Q that it is approximating, as shown later in this section.

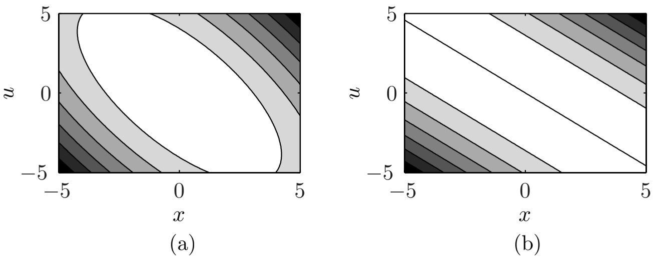

Fig. 2. Optimal value and advantage function for the example MDP in [16]. The system is $x_{k+1} = x_k + u_k$, using the optimal policy $\pi^*(x) = -Kx$ with K the state feedback solution based on the reward function $r_k = -x_k^2 - 0.1u_k^2$. The advantage function nicely shows the zero contour line of the optimal action $u = -Kx$. (a) Value function $Q^*(x,u)$. (b) Advantage function $A^*(x,u)$.

value of (23) is zero for the optimal w , i.e.,

$$\int_X d^\pi(x) \int_U \pi(x,u) \nabla_w \mathcal{E}_w^\pi(x,u) du dx = 0. \quad (24)$$

The policy gradient theorem with function approximation is based on the equality in (24).

Theorem 2 (Policy Gradient with Function Approximation): If h_w satisfies (24) and

$$\nabla_w h_w(x,u) = \nabla_\vartheta \ln \pi_\vartheta(x,u) \quad (25)$$

where $\pi_\vartheta(x,u)$ denotes the stochastic policy, parameterized by ϑ, then

$$\nabla_\vartheta J = \int_X d^\pi(x) \int_U \nabla_\vartheta \pi(x,u) h_w(x,u) du dx. \quad (26)$$

Proof: See [19]. ■

An extra assumption in [1] is that in (25), h actually needs to be an approximator that is linear with respect to some parameter w and features ψ, i.e., $h_w = w^\top \psi(x,u)$, transforming condition (25) into

$$\psi(x,u) = \nabla_\vartheta \ln \pi_\vartheta(x,u). \quad (27)$$

Features ψ that satisfy (27) are known as *compatible* features [1], [19], [51]. In the remainder of the paper, these will always be denoted by ψ and their corresponding parameters with w.

A technical issue, discussed in detail in [16] and [19], is that using the compatible function approximation $h_w = w^\top \nabla_\vartheta \ln \pi_\vartheta(x,u)$ gives

$$\int_U \pi_\vartheta(x,u) h_w(x,u) du = w^\top \nabla_\vartheta \underbrace{\int_U \pi_\vartheta(x,u) du}_{=1} = 0.$$

This shows that the expected value of $h_w(x,u)$ under the policy π_ϑ is zero for each state, from which it can be concluded that h_w is generally better thought of as the *advantage function* $A^\pi(x,u) = Q^\pi(x,u) - V^\pi(x)$. In essence, this means that using only compatible features for the value function results in an approximator that can only represent the relative value of an action u in some state x correctly, but not the absolute value $Q(x,u)$. An example showing how different the value function $Q(x,u)$ and the corresponding advantage function $A(x,u)$ can look is shown in Fig. 2. Because of this difference, the policy gradient estimate produced by just the compatible approximation will still have a high variance. To lower the variance, extra features have to be added on top of the compatible features, which take the role of modeling the difference between the advantage function $A^{\pi}(x,u)$ and the state-action value function $Q^{\pi}(x,u)$, i.e., the value function $V^{\pi}(x)$. These extra features are, therefore, only state-dependent, as dependence on the action would introduce a bias into the gradient estimate. The state-dependent offset that is created by these additional features is often referred to as a (reinforcement) baseline. The policy gradient theorem actually generalizes to the case where a state-dependent baseline function is taken into account. Equation (26) would then read

$$\nabla_{\vartheta} J = \int_X d^{\pi}(x) \int_U \nabla_{\vartheta}\pi(x,u)\left[h_w(x,u) + b(x)\right] du dx. \tag{28}$$

where $b(x)$ is the baseline function that can be chosen arbitrarily. Adding a baseline will not affect the unbiasedness of the gradient estimate, but can improve the accuracy of the critic's approximation and prevent an ill-conditioned projection of the value function on the compatible features ψ [1]. In that respect, this paper treats w as a subset of θ and ψ as a subset of ϕ. In practice, the optimal baseline, i.e., the baseline that minimizes the variance in the gradient estimate for the policy π, is the value function $V^{\pi}(x)$ [19], [20]. In [52], it is noted that, in light of the policy gradient theorem that was only published many years later, Gullapalli's earlier idea in [32] of using the TD δ in the gradient used to update the policy weights can be shown to yield the true policy gradient $\nabla_{\vartheta} J(\vartheta)$ and, hence, corresponds to the policy gradient theorem with respect to (28).

Theorem 2 yields a major benefit. Once a good parameterization for a policy has been found, a parameterization for the value function automatically follows and also guarantees convergence. Further on in this paper, many actor-critic algorithms make use of this theorem.

Part of this paper is dedicated to giving some examples of relevant actor-critic algorithms in both the standard gradient and natural gradient setting. As it is not possible to describe all existing actor-critic algorithms in this survey in detail, the algorithms addressed in this paper are chosen based on their originality: either they were the first to use a certain technique, extended an existing method significantly or the containing paper provided an essential analysis. In Section II, a distinction between the discounted and average reward setting was already made. The reward setting is the first major axis along which the algorithms in this paper are organized. The second major axis is the gradient type, which will be either the standard gradient or the natural gradient. This results in a total of four categories to which the algorithms can (uniquely) belong (see Table I). References in bold are discussed from an algorithmic perspective. Section IV describes actor-critic algorithms that use a standard gradient. Section V first introduces the concept of a natural gradient, after which natural actor-critic algorithms are discussed. References in italic are discussed in the Section VI on applications.

TABLE I
ACTOR-CRITIC METHODS, CATEGORIZED ALONG TWO AXES: GRADIENT TYPE AND REWARD SETTING

	Standard gradient	Natural gradient
Discounted return	**Barto et al. [46], FACRLN [53], [54], CACM [55], Bhatnagar [56]**, *Chun-Gui et al. [57], Kimura et al. [58], Raju et al. [59]*	***(e)NAC [16], [52]*, Park et al. [60], Girgin and Preux [61], Kimura [62]**,*Richter et al. [2], Kim et al. [63], Nakamura et al. [64], El-Fakdi et al. [65]*
Average return	**Konda and Tsitsiklis [1], *Paschalidis et al. [50]*, ACFRL [5], [66], Bhatnagar et al. I [20]**, *ACSMDP [67]*	**Bhatnagar et al. II–IV [20], gNAC [68]**

IV. Standard Gradient Actor-Critic Algorithms

Many papers refer to Barto *et al.* [46] as the starting point of actor-critic algorithms, although there the actor and critic were called the associative search element and adaptive critic element, respectively. That paper itself mentions that the implemented strategy is closely related to [45] and [69]. Either way, it is true that Barto *et al.* [46] defined the actor-critic structure that resembles the recently proposed actor-critic algorithms the most. Therefore, the discussion on standard actor-critic algorithms starts with this work, after which other algorithms in the discounted return setting are discussed. As many algorithms based on the average return also exist, they are dealt with in a separate section.

A. Discounted Return Setting

Barto *et al.* [46] use simple parameterizations

$$V_{\theta}(x) = \theta^{\top}\phi(x), \qquad \pi_{\vartheta}(x) = \vartheta^{\top}\phi(x)$$

with the same features $\phi(x)$ for the actor and critic. They chose binary features, i.e., for some state x only one feature $\phi_i(x)$ has a nonzero value, in this case $\phi_i(x) = 1$. For ease of notation, the state x was taken to be a vector of zeros with only one element equal to 1, indicating the activated feature. This allowed the parameterization to be written as

$$V_{\theta}(x) = \theta^{\top}x, \qquad \pi_{\vartheta}(x) = \vartheta^{\top}x.$$

Then, they were able to learn a solution to the well-known cart-pole problem using the update equations

$$\delta_k = r_{k+1} + \gamma V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k) \tag{29a}$$

$$z_{c,k} = \lambda_c z_{c,k-1} + (1-\lambda_c)x_k \tag{29b}$$

$$z_{a,k} = \lambda_a z_{a,k-1} + (1-\lambda_a)u_k x_k \tag{29c}$$

$$\theta_{k+1} = \theta_k + \alpha_c \delta_k z_{c,k} \tag{29d}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_a \delta_k z_{a,k} \tag{29e}$$

with

$$u_k = \tau\left(\pi_{\vartheta_k}(x_k) + n_k\right)$$

where τ is a threshold, sigmoid, or identity function, n_k is noise which accounts for exploration, and z_c, z_a are eligibility traces for the critic and actor, respectively. Note that these update equations are similar to the ones in template (19), considering the representation in binary features. The product $\delta_k z_{a,k}$ in (29e) can then be interpreted as the gradient of the performance with respect to the policy parameter.

Although no use was made of advanced function approximation techniques, good results were obtained. A mere division of the state space into boxes meant that there was no generalization among the states, indicating that learning speeds could definitely be improved upon. Nevertheless, the actor-critic structure itself remained and later work largely focused on better representations for the actor and the calculation of the critic.

Based on earlier work in [53], Wang *et al.* [54] introduced the fuzzy actor-critic reinforcement learning network (FACRLN), which uses only one fuzzy neural network based on radial basis functions for both the actor and the critic. That is, they both use the same input and hidden layers, but differ in their output by using different weights. This is based on the idea that both actor and critic have the same input and also depend on the same system dynamics. Apart from the regular updates to the actor and critic based on the TD error, the algorithm not only updates the parameters of the radial basis functions in the neural network, but also adaptively adds and merges fuzzy rules. Whenever the TD error or the squared TD error is too high and the so-called ϵ-completeness property [70] is violated, a new rule, established by a new radial basis function, is added to the network. A closeness measure of the radial basis functions decides whether two (or more) rules should be merged into one. For example, when using Gaussian functions in the network, if two rules have their centers and their widths close enough to each other, they will be merged into one. FACRLN is benchmarked against several other (fuzzy) actor-critic algorithms, including the original work in [46], and turns out to outperform them all in terms of the number of trials needed, without increasing the number of basis functions significantly.

At about the same time, Niedzwiedz *et al.* [55] also claimed, like with FACRLN, that there is redundancy in learning separate networks for the actor and critic and developed their consolidated actor-critic model (CACM) based on that same principle. They too set up a single neural network, using sigmoid functions instead of fuzzy rules, and use it for both the actor and the critic. The biggest difference is that here, the size of the neural network is fixed, i.e., there is no adaptive insertion/removal of sigmoid functions.

More recently, work by Bhatnagar on the use of actor-critic algorithms using function approximation for discounted cost MDPs under multiple inequality constraints appeared in [56]. The constraints considered are bounds on the expected values of discounted sums of single-stage cost functions ρ_n, i.e.,

$$S_n(\pi) = \sum_{x \in X} d_0(x) W_n^\pi(x) \leq s_n, \quad n = 1 \dots N$$

with

$$W_n^\pi(x) = E\left\{ \sum_{k=0}^{\infty} \gamma^k \rho_n(x_k, u_k) \middle| x_0 = x, \pi \right\}$$

and d_0 a given initial distribution over the states. The approach is, as in usual constrained optimization problems, to extend the discounted cost function $J(\pi)$ to a Lagrangian cost function

$$L(\pi, \bar{\mu}) = J(\pi) + \sum_{n=1}^{N} \mu_k G_n(\pi)$$

where $\bar{\mu} = (\mu_1, \dots, \mu_N)^\top$ is the vector of Lagrange multipliers, and $G_n(\pi) = S_n(\pi) - s_n$ are the functions representing the inequality constraints.

The algorithm generates an estimate of the policy gradient using simultaneous perturbation stochastic approximation (SPSA) [71], which has been found to be efficient even in high-dimensional parameter spaces. The SPSA gradient requires the introduction of two critics instead of one. The first critic, parameterized by $\theta^\top \phi(x)$, evaluates a policy parameterized by ϑ_k. The second critic, parameterized by $\theta'^\top \phi(x)$, evaluates a slightly perturbed policy parameterized by $\vartheta_k + \epsilon \Delta_k$ with a small $\epsilon > 0$. The element-wise policy parameter update is then given by[4]

$$\vartheta_{i,k+1} = \Gamma_i \left[\vartheta_k + \alpha_a \sum_{x \in X} d_0(x) \left(\frac{(\theta_k - \theta'_k)^\top \phi(x)}{\epsilon \Delta_{i,k}} \right) \right] \tag{30}$$

where Γ_i is a truncation operator. The Lagrange parameters μ also have an update rule of their own (further details in [56]), which introduces a third learning rate $\alpha_{L,k}$ into the algorithm for which the regular conditions

$$\sum_k \alpha_{L,k} = \infty, \qquad \sum_k \alpha_{L,k}^2 < \infty$$

must be satisfied and another constraint relating $\alpha_{L,k}$ to the actor step size $\alpha_{a,k}$

$$\lim_{k \to \infty} \frac{\alpha_{L,k}}{\alpha_{a,k}} = 0$$

must also hold, indicating that the learning rate for the Lagrange multipliers should decrease quicker than the actor's learning rate. Under these conditions, the authors prove the almost sure convergence to a locally optimal policy.

B. Average Reward Setting

In [1], together with the presentation of the novel ideas of compatible features, discussed in Section III-D, two actor-critic algorithms were introduced, differing only in the way they update the critic. The general update equations for these algorithms are

$$\widehat{J}_k = \widehat{J}_{k-1} + \alpha_{c,k}(r_{k+1} - \widehat{J}_{k-1}) \tag{31a}$$

$$\delta_k = r_{k+1} - \widehat{J}_k + Q_{\theta_k}(x_{k+1}, u_{k+1}) - Q_{\theta_k}(x_k, u_k) \tag{31b}$$

$$\theta_{k+1} = \theta_k + \alpha_{c,k} \delta_k z_k \tag{31c}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k} \Gamma(\theta_k) Q_{\theta_k}(x_k, u_k) \psi(x_k, u_k) \tag{31d}$$

where ψ is the vector of compatible features as defined in (27), and the parameterization Q_θ also contains these compatible

[4]This requires two simultaneous simulations of the constrained MDP.

features. The first and the second equation depict the standard update rules for the estimate of the average cost and the TD error. The third equation is the update of the critic. Here, the vector z_k represents an eligibility trace [7] and it is exactly this what distinguishes the two different algorithms described in the paper. The first algorithm uses a TD(1) critic, basically taking an eligibility trace with decay rate $\lambda = 1$. The eligibility trace is updated as

$$z_k = \begin{cases} z_{k-1} + \phi_k(x_k, u_k), & \text{if } x_k \neq x^S \\ \phi_k(x_k, u_k), & \text{otherwise} \end{cases}$$

where x^S is a special reset state for which it is assumed that the probability of reaching it from any initial state x within a finite number of transitions is bounded away from zero for any sequence of randomized stationary policies. Here, the eligibility trace is reset when encountering a state that meets this assumption. The second algorithm is a TD(λ) critic, simply updating the eligibility trace as

$$z_k = \lambda z_{k-1} + \phi_k(x_k, u_k).$$

The update of the actor in (31d) uses the policy gradient estimate from Theorem 2. It leaves out the state distribution $d^\pi(x)$ earlier seen in (26) of the policy gradient theorem, as the expected value of $\nabla J(\vartheta_k)$ is equal to that of $\nabla_\vartheta \pi(x,u)\widehat{Q}^\pi_w(x,u)$, and puts the critic's current estimate in place of $\widehat{Q}^\pi_w(x,u)$. Finally, $\Gamma(\theta_k)$ is a truncation term to control the step size of the actor, taking into account the current estimate of the critic. For this particular algorithm, some further assumptions on the truncation operator Γ must hold, which are not listed here.

It is known that using least-squares TD methods for policy evaluation is superior to using regular TD methods in terms of convergence rate as they are more data efficient [3], [47]. Inevitably, this resulted in work on actor-critic methods using an LSTD critic [52], [72]. However, Paschalidis *et al.* [50] showed that it is not straightforward to use LSTD without modification, as it undermines the assumptions on the step sizes (21), (22). As a result of the basics of LSTD, the step size schedule for the critic should be chosen as $\alpha_{c,k} = \frac{1}{k}$. Plugging this demand into (21) and (22) two conditions on the step size of the actor conflict, i.e.,

$$\sum_k \alpha_{a,k} = \infty, \qquad \sum_k (k\alpha_{a,k})^d < \infty, \qquad \text{for some } d > 0.$$

They conflict because the first requires α_a to decay at a rate slower than $1/k$, while the second demands a rate faster than $1/k$. This means there is a tradeoff between the actor having too much influence on the critic and the actor decreasing its learning rate too fast. The approach presented in [50] to address this problem is to use the following step size schedule for the actor. For some $K >> 1$, let $L = \lfloor k/K \rfloor$ and

$$\alpha_{a,k} := \frac{1}{L+1}\hat{\alpha}_a(k + 1 - LK)$$

where $\sum_k (k\hat{\alpha}_a(k))^d \leq \infty$ for some $d > 0$. As a possible example

$$\hat{\alpha}_{a,k}(b) := \varrho(C) \cdot b^{-C}$$

is provided, where $C > 1$ and $\varrho(C) > 0$. The critic's step size schedule is redefined as

$$\alpha_{c,k} := \frac{1}{k - \kappa(L,K)}.$$

Two extreme cases of $\kappa(L,K)$ are $\kappa(L,K) \overset{\triangle}{=} 0$ and $\kappa(L,K) = LK - 1$. The first alternative corresponds to the unmodified version of LSTD and the latter corresponds to "restarting" the LSTD procedure when k is an integer multiple of K. The reason for adding the κ term to the critic update is theoretical, as it may be used to increase the accuracy of the critic estimates for $k \to \infty$. Nevertheless, choosing $\kappa(L,K) = 0$ gave good results in the simulations in [50]. These step size schedules for the actor and critic allow the critic to converge to the policy gradient, despite the intermediate actor updates, while constantly reviving the learning rate of the actor such that the policy updates do not stop prematurely. The actor step size schedule does not meet the requirement $\sum_k (k\alpha_a)^d < \infty$ for some $d > 0$, meaning that convergence of the critic for the entire horizon cannot be directly established. What *is* proven by the authors is that the critic converges before every time instant $k = JK$, at which point a new epoch starts.[5] For the actor, the optimum is not reached during each epoch, but in the long run, it will move to an optimal policy. A detailed proof of this is provided in [50].

Berenji and Vengerov [5] used the analysis of [1] to provide a proof of convergence for an actor-critic fuzzy reinforcement learning (ACFRL) algorithm. The fuzzy element of the algorithm is the actor, which uses a parameterized fuzzy Takagi–Sugeno rulebase. The authors show that this parameterization adheres to the assumptions needed for convergence stated in [1], hence providing the convergence proof. The update equations for the average cost and the critic are the same as (31a) and (31c), but the actor update is slightly changed into

$$\vartheta_{k+1} = \Gamma\left(\vartheta_k + \alpha_{a,k}\theta_k^\top \phi_k(x_k,u_k)\psi_k(x_k,u_k)\right)$$

i.e., the truncation operator Γ is now acting on the complete update expression, instead of limiting the step size based on the critic's parameter. While applying ACFRL to a power management control problem, it was acknowledged that the highly stochastic nature of the problem and the presence of delayed rewards necessitated a slight adaptation to the original framework in [1]. The solution was to split the updating algorithm into three phases. Each phase consists of running a finite number of simulation traces. The first phase only estimates the average cost $\hat{J}$, keeping the actor and critic fixed. The second phase only updates the critic, based on the $\hat{J}$ obtained in the previous phase. This phase consists of a finite number of traces during which a fixed *positive* exploration term is used on top of the actor's output and an equal number of traces during which a fixed *negative* exploration term is used. The claim is that this systematic way of exploring is very beneficial in problems with delayed rewards, as it allows the critic to better establish the effects of a certain direction of exploration. The third and final phase keeps the critic fixed and lets the actor learn the new

[5]The authors use the term "episode," but this might cause confusion with the commonly seen concept of episodic tasks in RL, which is not the case here.

policy. Using this algorithm, ACFRL consistently converged to the same neighborhood of policy parameters for a given initial parameterization. Later, the authors extended the algorithm to ACFRL-2 in [66], which took the idea of systematic exploration one step further by learning two separate critics: one for positive exploration and one for negative exploration.

Bhatnagar *et al.* [20] introduced four algorithms. The first one uses a regular gradient and will, therefore, be discussed in this section. The update equations for this algorithm are

$$\widehat{J}_k = \widehat{J}_{k-1} + \alpha_{J,k}(r_{k+1} - \widehat{J}_{k-1}) \tag{32a}$$

$$\delta_k = r_{k+1} - \hat{J}_k + V_{\theta_k}(x_{k+1}) - V_{\theta_k}(x_k) \tag{32b}$$

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_k\phi(x_k) \tag{32c}$$

$$\vartheta_{k+1} = \Gamma(\vartheta_k + \alpha_{a,k}\delta_k\psi(x_k, u_k)). \tag{32d}$$

The critic update is simply an update in the direction of the gradient $\nabla_\theta V$. The actor update uses the fact that $\delta_k\psi(x_k, u_k)$ is an unbiased estimate of $\nabla_\vartheta J$ under conditions mentioned in [20]. The operator Γ is a projection operator, ensuring boundedness of the actor update. Three more algorithms are discussed in [20], but these make use of a natural gradient for the updates and hence are discussed in Section V-C2.

V. Natural Gradient Actor-Critic Algorithms

The previous section introduced actor-critic algorithms which use standard gradients. The use of standard gradients comes with drawbacks. Standard gradient descent is most useful for cost functions that have a single minimum and whose gradients are isotropic in magnitude with respect to any direction away from its minimum [73]. In practice, these two properties are almost never true. The existence of multiple local minima of the cost function, for example, is a known problem in RL, usually overcome by exploration strategies. Furthermore, the performance of methods that use standard gradients relies heavily on the choice of a coordinate system over which the cost function is defined. This noncovariance is one of the most important drawbacks of standard gradients [51], [74]. An example for this will be given later in this section.

In robotics, it is common to have a curved state space (manifold) because of the presence of angles in the state. A cost function will then usually be defined in that curved space too, possibly causing inefficient policy gradient updates to occur. This is exactly what makes the *natural* gradient interesting, as it incorporates knowledge about the curvature of the space into the gradient. It is a metric based not on the choice of coordinates, but on the manifold that those coordinates parameterize [51].

This section is divided into two parts. The first part explains what the concept of a natural gradient is and what its effects are in a simple optimization problem, i.e., not considering a learning setting. The second part is devoted to actor-critic algorithms that make use of this type of gradient to update the actor. As these policy updates are using natural gradients, these algorithms are also referred to as natural policy gradient algorithms.

A. *Natural Gradient in Optimization*

To introduce the notion of a natural gradient, this section summarizes work presented in [73]–[75]. Suppose a function $J(\vartheta)$ is parameterized by ϑ. When ϑ lives in a Euclidean space, the squared Euclidean norm of a small increment $\Delta\vartheta$ is given by the inner product

$$\|\Delta\vartheta\|_E^2 = \Delta\vartheta^\top\Delta\vartheta.$$

A steepest descent direction is then defined by minimizing $J(\vartheta + \Delta\vartheta)$, while keeping $\|\Delta\vartheta\|_E$ equal to a small constant. When ϑ is transformed to other coordinates $\widetilde{\vartheta}$ in a non-Euclidean space, the squared norm of a small increment $\Delta\widetilde{\vartheta}$ with respect to that *Riemannian* space is given by the product

$$\|\Delta\widetilde{\vartheta}\|_R^2 = \Delta\widetilde{\vartheta}^\top G(\widetilde{\vartheta})\Delta\widetilde{\vartheta}$$

where $G(\widetilde{\vartheta})$ is the Riemannian metric tensor, an $n \times n$ positive-definite matrix characterizing the intrinsic local curvature of a particular manifold in an n-dimensional space. The Riemannian metric tensor $G(\widetilde{\vartheta})$ can be determined from the relationship [73]:

$$\|\Delta\vartheta\|_E^2 = \|\Delta\widetilde{\vartheta}\|_R^2.$$

Clearly, for Euclidean spaces, $G(\widetilde{\vartheta})$ is the identity matrix.

Standard gradient descent for the new parameters $\widetilde{\vartheta}$ would define the steepest descent with respect to the norm $\|\Delta\widetilde{\vartheta}\|^2 = \Delta\widetilde{\vartheta}^\top\Delta\widetilde{\vartheta}$. However, this would result in a different gradient direction, despite keeping the same cost function and only changing the coordinates. The *natural* gradient avoids this problem, and always points in the "right" direction, by taking into account the Riemannian structure of the parameterized space over which the cost function is defined. So now, $\widetilde{J}(\widetilde{\vartheta} + \Delta\widetilde{\vartheta})$ is minimized while keeping $\|\Delta\widetilde{\vartheta}\|_R$ small ($\widetilde{J}$ here is just the original cost J, but written as a function of the new coordinates). This results in the natural gradient $\widetilde{\nabla}_{\widetilde{\vartheta}}\widetilde{J}(\widetilde{\vartheta})$ of the cost function, which is just a linear transformation of the standard gradient $\nabla_{\widetilde{\vartheta}}\widetilde{J}(\widetilde{\vartheta})$ by the inverse of $G(\widetilde{\vartheta})$:

$$\widetilde{\nabla}_{\widetilde{\vartheta}}\widetilde{J}(\widetilde{\vartheta}) = G^{-1}(\widetilde{\vartheta})\nabla_{\widetilde{\vartheta}}\widetilde{J}(\widetilde{\vartheta}).$$

As an example of optimization with a standard gradient versus a natural gradient, consider a cost function based on polar coordinates

$$J_P(r, \varphi) = \frac{1}{2}[(r\cos\varphi - 1)^2 + r^2\sin^2\varphi]. \tag{33}$$

This cost function is equivalent to $J_E(x, y) = (x - 1)^2 + y^2$, where x and y are Euclidean coordinates, i.e., the relationship between (r, φ) and (x, y) is given by

$$x = r\cos\varphi, \qquad y = r\sin\varphi.$$

Fig. 3(a) shows the contours and antigradients of $J_P(r, \varphi)$ for $0 \le r \le 3$ and $|\varphi| \le \pi$, where

$$-\nabla_{(r,\varphi)}J_P(r, \varphi) = -\begin{bmatrix} r - \cos\varphi \\ r\sin\varphi \end{bmatrix}.$$

The magnitude of the gradient clearly varies widely over the (r, φ)-plane. When performing a steepest descent search on this

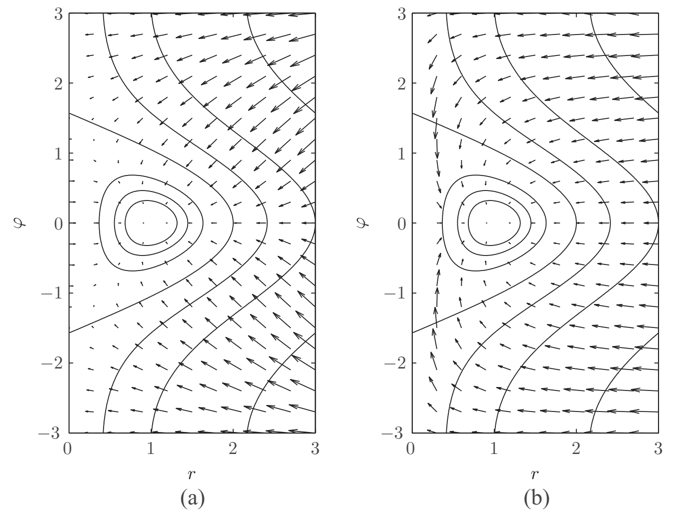

Fig. 3. (a) Standard and (b) natural gradients of the cost function $J_P(r,\varphi)$ in polar coordinates.

cost function, the trajectories from any point (r,φ) to an optimal one will be far from straight paths. For the transformation of Euclidean coordinates into polar coordinates, the Riemannian metric tensor is [73]

$$G(r,\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & r^2 \end{bmatrix}$$

so that the natural gradient of the cost function in (33) is

$$\begin{aligned} -\widetilde{\nabla}_{(r,\varphi)} J_P(r,\varphi) &= -G(r,\varphi)^{-1} \nabla_{(r,\varphi)} J_P(r,\varphi) \\ &= -\begin{bmatrix} r - \cos\varphi \\ \dfrac{\sin\varphi}{r} \end{bmatrix}. \end{aligned}$$

Fig. 3(b) shows the natural gradients of $J_P(r,\varphi)$. Clearly, the magnitude of the gradient is now more uniform across the space, and the angles of the gradients also do not greatly vary away from the optimal point (1,0).

Fig. 4 shows the difference between a steepest descent method using a standard gradient and a natural gradient on the cost $J_P(r,\varphi)$ using a number of different initial conditions. The natural gradient clearly performs better as it always finds the optimal point, whereas the standard gradient generates paths that are leading to points in the space which are not even feasible, because of the radius which needs to be positive.

To get an intuitive understanding of what the effect of a natural gradient is, Fig. 5 shows trajectories for the standard and natural gradient that have been transformed onto the Euclidean space. Whatever the initial condition[6] is, the natural gradient of $J_P(r,\varphi)$ always points straight to the optimum and follows the same path that the standard gradient of $J_E(x,y)$ would do.

When $J(\vartheta)$ is a quadratic function of ϑ (like in many optimization problems, including for example those solved in supervised learning), the Hessian $H(\vartheta)$ is equal to $G(\vartheta)$ for the underlying parameter space, and there is no difference between using

[6]The exemplified initial conditions are not the same as in Fig. 4.

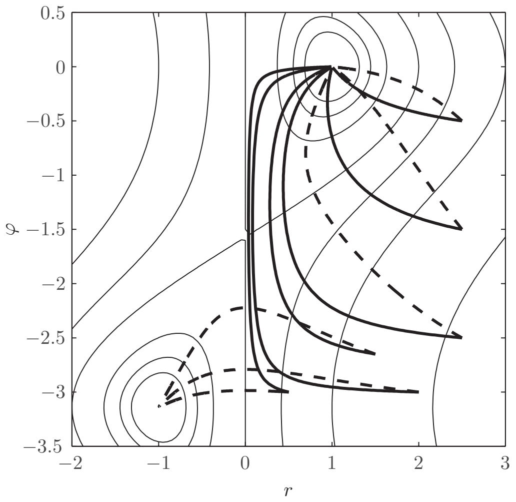

Fig. 4. Trajectories for standard gradient (dashed) and natural gradient (solid) algorithms for minimizing $J_P(r,\varphi)$ in polar coordinates.

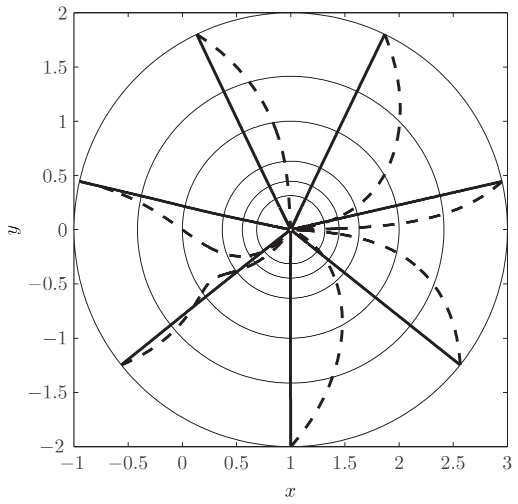

Fig. 5. Trajectories for standard gradient (dashed) and natural gradient (solid) algorithms for minimizing $J_P(r,\varphi)$, transformed to Euclidean coordinates.

Newton's method and natural gradient optimization. In general however, natural gradient optimization differs from Newton's method, e.g., $G(\vartheta)$ is always positive definite by construction, whereas the Hessian $H(\vartheta)$ may not be [73]. The general intuition developed in this section is essential before moving on to the natural policy gradient in MDPs, explained next.

B. Natural Policy Gradient

The possibility of using natural gradients in online learning was first appreciated in [75]. As shown above, the crucial property of the natural gradient is that it takes into account the structure of the manifold over which the cost function is defined, locally characterized by the Riemannian metric tensor. To apply this insight in the context of *policy* gradient methods, the main question is then what is an appropriate manifold, and once that is known, what is its Riemannian metric tensor.

Consider first just the parameterized stochastic policy $\pi_\vartheta(x, u)$ at a single state x, a probability distribution over the actions u. This policy is a point on a manifold of such probability distributions, found at coordinates ϑ. For a manifold of distributions, the Riemannian tensor is the so-called Fisher information matrix (FIM) [75], which for the policy above is [51]

$$\begin{aligned} F(\vartheta, x) &= \mathrm{E}\left[\nabla_\vartheta \ln \pi_\vartheta(x,u) \nabla_\vartheta \ln \pi_\vartheta(x,u)^\top\right] \\ &= \int_U \pi_\vartheta(x,u) \nabla_\vartheta \ln \pi_\vartheta(x,u) \nabla_\vartheta \ln \pi_\vartheta(x,u)^\top du. \quad (34) \end{aligned}$$

The single-state policy is directly related with the expected immediate reward, over a single step from x. However, it does not tell much about the overall expected return $J(\pi)$, which is defined over entire state trajectories. To obtain an appropriate overall FIM, in the average reward case, Kakade [51] made the intuitive choice of taking the expectation of $F(\vartheta, x)$ with respect to the stationary state distribution $d^\pi(x)$

$$F(\vartheta) = \int_X d^\pi(x) F(\vartheta, x) dx. \quad (35)$$

He was, however, unsure whether this was the right choice.

Later on, the authors of [52] and [74] independently showed that (35) is indeed a true FIM, for the manifold of *probability distributions over trajectories* in the MDP. When used to control the MDP with stochastic dynamics f, $\pi_\vartheta(x, u)$ gives rise to different controlled trajectories with different probabilities; therefore, each value of the parameter ϑ yields such a distribution over trajectories. To understand how this distribution is relevant to the value $J(\pi)$ of the policy, observe that this value can be written as the expected value of the infinite-horizon return over all possible paths, where the expectation is taken with respect to precisely the trajectory distribution. Furthermore, in [52] and [74], the authors show that this idea also extends to the discounted reward case, where the FIM is still given by (35), only with $d^\pi(x)$ replaced by the discounted state distribution $d^\pi_\gamma(x)$, as defined in Section II-A.

Examples of the difference in performance between regular policy gradients and natural policy gradients are provided in [51], [52], [74].

C. Natural Actor-Critic Algorithms

This section describes several representative actor-critic algorithms that employ a natural policy gradient. Again, a distinction is made between algorithms using the discounted return and the average return.

1) Discounted Return Setting: After the acknowledgement by Amari [75] that using the natural gradient could be beneficial for learning, the aptly called natural actor-critic algorithm in [52] was, to the best of our knowledge, the first actor-critic algorithm that successfully employed a natural gradient for the policy updates. Together with [51], they gave a proof that the natural gradient $\widetilde{\nabla}_\vartheta J(\vartheta)$ is in fact the compatible feature parameter w of the approximated value function, i.e.,

$$\widetilde{\nabla}_\vartheta J(\vartheta) = w.$$

Consequently, they were able to use a natural gradient without explicitly calculating the FIM. This turns the policy update step into

$$\begin{aligned} \vartheta_{k+1} &= \vartheta_k + \alpha_a \widetilde{\nabla}_\vartheta J(\vartheta) \quad (36a) \\ &= \vartheta_k + \alpha_a w_{k+1}. \quad (36b) \end{aligned}$$

For the policy evaluation step of the algorithm, i.e., the calculation of the critic parameter w, LSTD-Q(λ) was used, which was their own extension to LSTD(λ) from [3]. The natural actor-critic outperformed standard gradient policy gradient methods on a cart-pole balancing setup. Later, the work was extended in [16], where it was shown that several well-known reinforcement algorithms (e.g., Sutton and Barto's actor-critic [7] and Bradtke's Q-learning [23]) are strongly related to natural actor-critic algorithms. Furthermore, the paper presents the successful application of an episodic variant of natural actor-critic (eNAC) on an anthropomorphic robot arm. For another example of a natural-actor critic algorithm with a regression-based critic, see [76].

Park *et al.* [60] extend the original work in [52] by using a recursive least-squares method in the critic, making the parameter estimation of the critic more efficient. They then successfully apply it to the control of a two-link robot arm.

Girgin and Preux [61] improve the performance of natural actor-critic algorithms, by using a neural network for the actor, which includes a mechanism to automatically add hidden layers to the neural network if the accuracy is not sufficient. Enhancing the eNAC method in [16] with this basis expansion method clearly showed its benefits on a cart-pole simulation.

Although a lot of (natural) actor-critic algorithms use sophisticated function approximators, Kimura showed in [62] that a simple policy parameterization using rectangular coarse coding can still outperform conventional Q-learning in high-dimensional problems. In the simulations, however, Q-learning did outperform the natural actor-critic algorithm in low-dimensional problems.

2) Average reward setting: Bhatnagar *et al.* [20] introduced four algorithms, three of which are natural-gradient algorithms. They extend the results of [1] by using TD learning for the actor and by incorporating natural gradients. They also extend the work of [16] by providing the first convergence proofs and the first fully incremental natural actor-critic algorithms. The contribution of convergence proofs for natural-actor critic algorithms is important, especially since the algorithms utilized both function approximation and a bootstrapping critic, a combination which is essential to large-scale applications of RL. The second algorithm only differs from the first algorithm, described at the end of Section IV-B with (32), in the actor update (32d). It directly substitutes the standard gradient with the natural gradient

$$\vartheta_{k+1} = \Gamma(\vartheta_k + \alpha_{a,k} F_k^{-1}(\vartheta) \delta_k \psi(x_k, u_k)) \quad (37)$$

where F is the FIM. This requires the actual calculation of the FIM. Since the FIM can be written using the compatible features

ψ as

$$F(\vartheta) = \int_X d^\pi(x) \int_U \pi(x,u)\psi(x,u)\psi^\top(x,u)dudx$$

sample averages can be used to compute it

$$F_k(\vartheta) = \frac{1}{k+1}\sum_{l=0}^{k}\psi(x_l,u_l)\psi^\top(x_l,u_l).$$

After converting this equation to a recursive update rule, and putting the critic's learning rate in place, the Sherman–Morrison matrix inversion lemma is used to obtain an iterative update rule for the inverse of the FIM[7]

$$F_k^{-1}(\vartheta) = \frac{1}{1-\alpha_{c,k}} \cdot \left[F_{k-1}^{-1} - \alpha_{c,k}\frac{(F_{k-1}^{-1}\psi_k)(F_{k-1}^{-1}\psi_k)^\top}{1-\alpha_{c,k}(1-\psi_k^\top F_{k-1}^{-1}\psi_k)}\right]$$

where the initial value F_0^{-1} is chosen to be a scalar multiple of the identity matrix. This update rule, together with the adjusted update of the actor, then form the second algorithm.

The third algorithm in [20] uses the fact that the compatible approximation $w^\top\psi(x,u)$ is better thought of as an advantage function approximator instead of a state-action value function approximator, as mentioned in Section III-D. Hence, the algorithm tunes the weights w, such that the squared error $\mathcal{E}^\pi(w) = E\left[(w^\top\psi(x,u) - A^\pi(x,u))^2\right]$ is minimized. The gradient of this error is

$$\nabla_w\mathcal{E}^\pi(w) = 2\sum_X d^\pi(x)\sum_U \pi(x,u) \cdot \left[w^\top\psi(x,u) - A^\pi(x,u)\right]\psi(x,u).$$

As δ_k is an unbiased estimate of $A^\pi(x_k,u_k)$ (see [77]), the gradient is estimated with

$$\widehat{\nabla_w\mathcal{E}^\pi}(w) = 2(\psi_k\psi_k^\top w - \delta_k\psi_k) \tag{38}$$

and the gradient descent update rule for w (using the same learning rate as the critic) is

$$w_{k+1} = w_k - \alpha_{c,k}(\psi_k\psi_k^\top w_k - \delta_k\psi_k). \tag{39}$$

Furthermore, the natural gradient estimate is given by w (as shown by Peters and Schaal [16]), and an explicit calculation for the FIM is no longer necessary. Therefore, the third algorithm is obtained by using (39) and replacing the actor update in (37) with

$$\vartheta_{k+1} = \Gamma(\vartheta_k + \alpha_{a,k}w_{k+1}). \tag{40}$$

The fourth algorithm in [20] is obtained by combining the second and third algorithms. The explicit calculation of F_k^{-1} is now used for the update of the compatible parameter w. The update of w now also follows its natural gradient, by premultiplying the result in (38) with F_k^{-1}, i.e.,

$$\widehat{\widetilde{\nabla}_w\mathcal{E}^\pi}(w) = 2F_k^{-1}(\psi_k\psi_k^\top w - \delta_k\psi_k)$$

[7]For readability, $\psi(x_k,u_k)$ is replaced by ψ_k for the remainder of this section.

turning the update of w into

$$\begin{aligned} w_{k+1} &= w_k - \alpha_{c,k}F_k^{-1}(\psi_k\psi_k^\top w_k - \delta_k\psi_k) \\ &= w_k - \alpha_{c,k}\underbrace{F_k^{-1}\psi_k\psi_k^\top}_{I}w_k + \alpha_{c,k}F_k^{-1}\delta_k\psi_k \\ &= w_k - \alpha_{c,k}w_k + \alpha_{c,k}F_k^{-1}\delta_k\psi_k \end{aligned}$$

where clever use is made of the fact that F_k is written as the squared ψ's. The actor update is still (40).

Although most natural actor-critic algorithms use the natural gradient as defined in Section V, the generalized natural actor-critic (gNAC) algorithm in [68] does not. Instead, a *generalized* natural gradient (gNG) is used, which combines properties of the FIM and natural gradient as defined before with the properties of a differently defined FIM and natural gradient from the work in [78]. They consider the fact that the average reward $J(\vartheta)$ is not only affected by the policy π, but also by the resulting state distribution $d^\pi(x)$ and define the FIM of the state-action joint distribution as

$$F_{\mathrm{SA}}(\vartheta) = F_{\mathrm{S}}(\vartheta) + F_{\mathrm{A}}(\vartheta) \tag{41}$$

where $F_{\mathrm{S}}(\vartheta)$ is the FIM of the stationary state distribution $d^\pi(x)$ and $F_{\mathrm{A}}(\vartheta)$ the FIM as defined in (35). In [78], the use of $F_{\mathrm{SA}}(\vartheta)$ as the FIM is considered better for learning than using the original FIM because of three reasons:

1) Learning with $F_{\mathrm{SA}}(\vartheta)$ still benefits from the concepts of natural gradient, since it necessarily and sufficiently accounts for the probability distributions that the average reward depends on.
2) $F_{\mathrm{SA}}(\vartheta)$ is analogous to the Hessian matrix of the average reward.
3) Numerical experiments have shown a strong tendency of avoiding plateaus in learning.

Nevertheless, the original FIM $F_{\mathrm{A}}(\vartheta)$ accounts for the distribution over an infinite amount of time steps, whereas $F_{\mathrm{SA}}(\vartheta)$ only accounts for the distribution over a single time step. This might increase the mixing time of the Markov chain drastically, making it hard for the RL learning agent to estimate a gradient with a few samples. Therefore, the authors suggest to use a weighted average, using a weighting factor ι, of both FIMs defined in (34) and (41). The gNG is then calculated by using the inverse of this weighted average, leading to the policy gradient

$$\widetilde{\widetilde{\nabla}}_\vartheta J(\vartheta) = (\iota F_{\mathrm{S}} + F_{\mathrm{A}})^{-1}\nabla_\vartheta J(\vartheta).$$

The implementation of the algorithm is similar to that of NAC, with the slight difference that another algorithm, $\mathcal{LS}$LSD [79], is used to estimate $\nabla_\vartheta d^\pi(x)$. If $\iota = 0$, gNAC is equivalent to the original NAC algorithm of Peters *et al.* [52], but now optimizing over the average return instead of the discounted return. In a numerical experiment with a randomly synthesized MDP of 30 states and 2 actions, gNAC with $\iota > 0$ outperformed the original NAC algorithm.

VI. Applications

This section provides references to papers that have applied actor-critic algorithms in several domains. Note that the list of

Fig. 6. Episodic natural actor-critic method in [16] applied to an anthropomorphic robot arm performing a baseball bat swing task.

applications is not exhaustive and that other application domains for actor-critic algorithms and more literature on the applications mentioned below exists.

In the field of robotics, early successful results of using actor-critic type methods on real hardware were shown on a ball on a beam setup [80], a peg-in-hole insertion task [81], and biped locomotion [82]. Peters and Schaal showed in [16] that their natural actor-critic method was capable of getting an anthropomorphic robot arm to learn certain motor skills (see Fig. 6). Kim *et al.* [63] recently successfully applied a modified version of the algorithm in [60] to motor skill learning. Locomotion of a two-link robot arm was learned using a recursive least-squares natural actor-critic method in [60]. Another successful application on a real four-legged robot is given in [58]. Nakamura *et al.* devised an algorithm based on [16] which made a biped robot walk stably [64]. Underwater cable tracking [65] was done using the NAC method of [16], where it was used in both a simulation and real-time setting: Once the results from simulation were satisfactory, the policy was moved to an actual underwater vehicle, which continued learning during operation, improving the initial policy obtained from simulation.

An example of a logistics problem solved by actor-critic methods is given in [50], which successfully applies such a method to the problem of dispatching forklifts in a warehouse. This is a high-dimensional problem because of the number of products, forklifts, and depots involved. Even with over 200 million discrete states, the algorithm was able to converge to a solution that performed 20% better in terms of cost than a heuristic solution obtained by taking the exact solution of a smaller problem and expanding this to a large state space.

Usaha and Barria [67] use the algorithm from [1] described in Section IV-B, extended to handle semi-MDPs,[8] for call admission control in lower earth orbit satellite networks. They compared the performance of this actor-critic semi-Markov decision algorithm (ACSMDP) together with an optimistic policy iteration (OPI) method to an existing routing algorithm. While both ACSMDP and OPI outperform the existing routing algorithm, ACSMDP has an advantage in terms of computational time, although OPI reaches the best result. Based on the FACRLN from [54] in Section IV-A, Li *et al.* [57] devised a way to control traffic signals at an intersection and showed in simulation that this method outperformed the commonly seen time slice allocation methods. Richter *et al.* [2] showed similar improvements in road traffic optimization when using natural actor-critic methods.

[8]Semi-MDPs extend regular MDPs by taking into account a (possibly stochastically) varying transition time from one state to another.

Finally, an application to the finance domain was described in [59], where older work on actor-critic algorithms [83] was applied in the problem of determining dynamic prices in an electronic retail market.

VII. Discussion and Outlook

When applying RL to a certain problem, knowing *a priori* whether a critic-only, actor-only, or actor-critic algorithm will yield the best control policy is virtually impossible. However, a few rules of thumb should help in selecting the most sensible class of algorithms to use. The most important thing to consider first is the type of control policy that should be learned. If it is necessary for the control policy to produce actions in a continuous space, critic-only algorithms are no longer an option, as calculating a control law would require solving the possibly non-convex optimization procedure of (11) over the continuous action space. Conversely, when the controller only needs to generate actions in a (small) countable, finite space, it makes sense to use critic-only methods, as (11) can be solved by enumeration. Using a critic-only method also overcomes the problem of high-variance gradients in actor-only methods and the introduction of more tuning parameters (e.g., extra learning rates) in actor-critic methods.

Choosing between actor-only and actor-critic methods is more straightforward. If the problem is modeled by a (quasi-)stationary MDP, actor-critic methods should provide policy gradients with lower variance than actor-only methods. Actor-only methods are, however, more resilient to fast changing nonstationary environments, in which a critic would be incapable of keeping up with the time-varying nature of the process and would not provide useful information to the actor, canceling the advantages of using actor-critic algorithms. In summary, actor-critic algorithms are most sensibly used in a (quasi-)stationary setting with a continuous state and action space.

Once the choice for actor-critic has been made, the issue of choosing the right features for the actor and critic, respectively, remains. There is consensus, although, that the features for the actor and critic do not have to be chosen independently. Several actor-critic algorithms use the exact same set of features for both the actor and the critic, while the policy gradient theorem indicates that it is best to first choose a parameterization for the actor, after which *compatible features* for the critic can be derived. In this sense, the use of compatible features is beneficial as it *lessens* the burden of choosing a separate parameterization for the value function. Note that compatible features do not *eliminate* the burden of choosing features for the value function completely (see Section III-D). Adding state-dependent features to the value function on top of the compatible features remains an important task as this is the only way to further reduce the variance in the policy gradient estimates. How to choose these additional features remains a difficult problem.

Choosing a good parameterization for the policy in the first place also remains an important issue as it highly influences the performance after learning. Choosing this parameterization does seem less difficult than for the value function, as in practice it is easier to get an idea what shape the policy has than the corresponding value function.

One of the conditions for successful application of RL in practice is that learning should be quick. Although this paper focuses on gradient-based algorithms and how to estimate this gradient, it should be noted that it is not only the quality of the gradient estimate that influences the speed of learning. Balancing the exploration and exploitation of a policy and choosing good learning rate schedules also have a large effect on this, although more recently expectation-maximization methods that work without learning rates have been proposed [84], [85]. With respect to gradient type, the natural gradient seems to be superior to the standard gradient. However, an example of standard Q-learning on low-dimensional problems in [62] and relative entropy policy search [44] showed better results than the natural gradient. Hence, even though the field of natural gradient actor-critic methods is still a very promising area for future research, it does not always show superior performance compared with other methods. A number of applications which use natural gradients are mentioned in this paper. The use of compatible features makes it straightforward to calculate approximations of natural gradients, which implies that any actor-critic algorithm developed in the future should attempt to use this type of gradient, as it speeds up learning without any real additional computational effort.

ACKNOWLEDGMENT

The authors are grateful for the very helpful comments and suggestions that were received during the reviewing process.

REFERENCES

[1] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, 2003.

[2] S. Richter, D. Aberdeen, and J. Yu, "Natural actor-critic for road traffic optimisation," in *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007, pp. 1169–1176.

[3] J. A. Boyan, "Technical update: Least-squares temporal difference learning," *Mach. Learn.*, vol. 49, pp. 233–246, 2002.

[4] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Intell. Res.*, vol. 15, pp. 319–350, 2001.

[5] H. R. Berenji and D. Vengerov, "A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters," *IEEE Trans. Fuzzy Syst.*, vol. 11, no. 4, pp. 478–485, Aug. 2003.

[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

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

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[9] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS J. Comput.*, vol. 21, no. 2, pp. 178–192, 2009.

[10] C. Szepesvári, "Algorithms for reinforcement learning," in *Synthesis Lectures on Artificial Intelligence and Machine Learning*. New York: Morgan & Claypool, 2010.

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

[12] 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.

[13] P. Pennesi and I. C. Paschalidis, "A distributed actor-critic algorithm and applications to mobile sensor network coordination problems," *IEEE Trans. Autom. Control*, vol. 55, no. 2, pp. 492–497, Feb. 2010.

[14] C. Li, M. Wang, and Q. Yuan, "A multi-agent reinforcement learning using actor-critic methods," in *Proc. 7th Int. Conf. Mach. Learn. Cybern.*, Kunming, China, 2008, pp. 878–882.

[15] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2010.

[16] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, pp. 1180–1190, 2008.

[17] V. S. Borkar, "A sensitivity formula for risk-sensitive cost and the actor-critic algorithm," *Syst. Control Lett.*, vol. 44, no. 5, pp. 339–346, 2001.

[18] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Volume II*, 3rd ed. ed. Nashua, NH: Athena Scientific, 2007.

[19] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*. Cambridge, MA: MIT Press, 2000, pp. 1057–1063.

[20] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[21] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Univ. Cambridge, Cambridge, U.K., 1989

[22] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.

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

[24] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Univ. Cambridge, Cambridge, U.K., Tech. Rep. CUED/F-INFENG/TR 166, 1994

[25] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Mach. Learn.*, Tahoe City, CA, 1995, pp. 30–37.

[26] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proc. 12th Int. Conf. Mach. Learn.*, Tahoe City, CA, 1995, pp. 261–268.

[27] J. N. Tsitsiklis and B. Van Roy, "Feature-based methods for large scale dynamic programming," *Mach. Learn.*, vol. 22, pp. 59–94, 1996.

[28] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[29] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, 2008, pp. 664–671.

[30] R. Schoknecht, "Optimality of reinforcement learning algorithms with linear function approximation," in *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press, 2003, pp. 1555–1562.

[31] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Finite-sample analysis of LSTD," in *Proc. 27th Int. Conf. Mach. Learn.*, Haifa, Israel, 2010, pp. 615–622.

[32] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Netw.*, vol. 3, no. 6, pp. 671–692, 1990.

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.

[34] J. A. Bagnell and J. Schneider, "Policy search in kernel Hilbert space," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. 80, 2003.

[35] K. Kersting and K. Driessens, "Non-parametric policy gradients: A unified treatment of propositional and relational domains," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, 2008, pp. 456–463.

[36] V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva, "Stochastic optimization," *Eng. Cybern.*, vol. 5, pp. 11–16, 1968.

[37] P. W. Glynn, "Likelihood ratio gradient estimation: An overview," in *Proc. Winter Simul. Conf.*. Atlanta, GA: ACM Press, 1987, pp. 366–375.

[38] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Netw.*, vol. 21, no. 4, pp. 682–697, 2008.

[39] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming* (ser. Modern Analytic and Computational Methods in Science and Mathematics vol. 24). New York: American Elsevier, 1970.

[40] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control* (ser. Mathematics in Science and Engineering vol. 65). New York: Academic, 1970.

[41] L. Hasdorff, *Gradient Optimization and Nonlinear Control*. New York: Wiley, 1976.

[42] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proc. 28th Int. Conf. Mach. Learn.*, Bellevue, WA, 2011, pp. 465–472.

[43] M. Riedmiller, J. Peters, and S. Schaal, "Evaluation of policy gradient methods and variants on the cart-pole benchmark," in *Proc. IEEE Symp. Approx. Dyn. Programm. Reinforcement Learn.*, Honolulu, HI, 2007, pp. 254–261.

[44] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *Proc. 24th AAAI Conf. Artif. Intell.*, Atlanta, GA, 2010, pp. 1607–1612.

[45] I. H. Witten, "An adaptive optimal controller for discrete-time Markov environments," *Inf. Control*, vol. 34, pp. 286–295, 1977.

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

[47] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Mach. Learn.*, vol. 22, pp. 33–57, 1996.

[48] L. Baird and A. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press, 1999.

[49] J. N. Tsitsiklis and B. Van Roy, "Average cost temporal-difference learning," *Automatica*, vol. 35, no. 11, pp. 1799–1808, 1999.

[50] I. C. Paschalidis, K. Li, and R. M. Estanjini, "An actor-critic method using least squares temporal difference learning," in *Proc. Joint 48th IEEE Congr. Decis. Control/28th Chin. Control Conf.*, Shanghai, China, 2009, pp. 2564–2569.

[51] S. Kakade, "Natural policy gradient," in *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press, 2001, pp. 1531–1538.

[52] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," presented at the 3rd IEEE-RAS Int. Conf. Human. Robots, Karlsruhe, Germany, 2003

[53] Y. Cheng, J. Yi, and D. Zhao, "Application of actor-critic learning to adaptive state space construction," in *Proc. 3rd Int. Conf. Mach. Learn. Cybern.*, Shanghai, China, 2004, pp. 2985–2990.

[54] X. Wang, Y. Cheng, and J. Yi, "A fuzzy actor-critic reinforcement learning network," *Inf. Sci.*, vol. 177, pp. 3764–3781, 2007.

[55] C. Niedzwiedz, I. Elhanany, Z. Liu, and S. Livingston, "A consolidated actor-critic model with function approximation for high-dimensional POMDPs," in *Proc. AAAI Workshop Adv. POMDP Solvers*, Chicago, IL, 2008, pp. 37–42.

[56] S. Bhatnagar, "An actor-critic algorithm with function approximation for discounted cost constrained Markov decision processes," *Syst. Control Lett.*, vol. 59, no. 12, pp. 760–766, 2010.

[57] C. Li, M. Wang, Z. Sun, and Z. Zhang, "Urban traffic signal learning control using fuzzy actor-critic methods," in *Proc. 5th Int. Conf. Natural Comput.*, Tianjin, China, 2009, pp. 368–372.

[58] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in *Proc. 40th IEEE Conf. Decis. Control*, Orlando, FL, 2001, pp. 411–416.

[59] C. Raju, Y. Narahari, and K. Ravikumar, "Reinforcement learning applications in dynamic pricing of retail markets," in *Proc. IEEE Int. Conf. E-Commerce*, Newport Beach, CA, 2003, pp. 339–346.

[60] J. Park, J. Kim, and D. Kang, "An RLS-based natural actor-critic algorithm for locomotion of a two-linked robot arm," in *Lecture Notes on Artificial Intelligence 3801*. Berlin/Heidelberg, Germany: Springer-Verlag, 2005, pp. 65–72.

[61] S. Girgin and P. Preux, "Basis expansion in natural actor critic methods," in *Lecture Notes in Artificial Intelligence 5323*. Berlin, Germany: Springer-Verlag, 2008, pp. 110–123.

[62] H. Kimura, "Natural gradient actor-critic algorithms using random rectangular coarse coding," in *Proc. SICE Annu. Conf.*, Chofu, Japan, 2008, pp. 2027–2034.

[63] B. Kim, J. Park, S. Park, and S. Kang, "Impedance learning for robotic contact tasks using natural actor-critic algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 2, pp. 433–443, Apr. 2010.

[64] Y. Nakamura, T. Mori, M.-A. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a CPG-actor-critic method," *Neural Netw.*, vol. 20, pp. 723–735, 2007.

[65] A. El-Fakdi and E. Galceran, "Two steps natural actor critic learning for underwater cable tracking," in *Proc. IEEE Int. Conf. Robot. Autom.*, Anchorage, AK, 2010, pp. 2267–2272.

[66] D. Vengerov, N. Bambos, and H. R. Berenji, "A fuzzy reinforcement learning approach to power control in wireless transmitters," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 4, pp. 768–778, Aug. 2005.

[67] W. Usaha and J. A. Barria, "Reinforcement learning for resource allocation in LEO satellite networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 3, pp. 515–527, Jun. 2007.

[68] T. Morimura, E. Uchibe, J. Yoshimoto, and K. Doya, "A generalized natural actor-critic algorithm," in *Advances in Neural Information Processing Systems 22*. Cambridge, MA: MIT Press, 2009, pp. 1312–1320.

[69] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 211–229, 1959.

[70] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Part I," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–418, Mar./Apr. 1990.

[71] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Trans. Autom. Control*, vol. 37, no. 3, pp. 332–341, Mar. 1992.

[72] J. L. Williams, J. W. Fisher III, and A. S. Willsky, "Importance sampling actor-critic algorithms," in *Proc. Amer. Control Conf.*, Minneapolis, MN, 2006, pp. 1625–1630.

[73] S. Amari and S. C. Douglas, "Why natural gradient?," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Seattle, WA, 1998, pp. 1213–1216.

[74] J. A. Bagnell and J. Schneider, "Covariant policy search," in *Proc. 18th Int. Joint Conf. Artif. Intell.*, Acapulco, Mexico, 2003, pp. 1019–1024.

[75] S. Amari, "Natural gradient works efficiently in learning," *Neural Comput.*, vol. 10, no. 2, pp. 251–276, 1998.

[76] F. S. Melo and M. Lopes, "Fitted natural actor-critic: A new algorithm for continuous state-action MDPs," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, Antwerp, Belgium, 2008, pp. 66–81.

[77] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental natural actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 105–112.

[78] T. Morimura, E. Uchibe, J. Yoshimoto, and K. Doya, "A new natural policy gradient by stationary distribution metric," in *Lecture Notes in Artificial Intelligence 5212*. Berlin, Germany: Springer-Verlag, 2008, pp. 82–97.

[79] T. Morimura, E. Uchibe, J. Yoshimoto, J. Peters, and K. Doya, "Derivatives of logarithmic stationary distributions for policy gradient reinforcement learning," *Neural Comput.*, vol. 22, no. 2, pp. 342–376, 2010.

[80] H. Benbrahim, J. Doleac, J. A. Franklin, and O. Selfridge, "Real-time learning: A ball on a beam," in *Proc. Int. Joint Conf. Neural Netw.*, Baltimore, MD, 1992, pp. 98–103.

[81] V. Gullapalli, "Learning control under extreme uncertainty," in *Advances in Neural Information Processing Systems 5*. San Mateo, CA: Morgan Kaufmann, 1993, pp. 327–334.

[82] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robot. Auton. Syst.*, vol. 22, pp. 283–302, 1997.

[83] V. R. Konda and V. S. Borkar, "Actor-critic–type learning algorithms for Markov decision processes," *SIAM J. Control Optim.*, vol. 38, no. 1, pp. 94–123, 1999.

[84] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Mach. Learn.*, vol. 84, pp. 171–203, 2011.

[85] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis, "Learning model-free robot control by a Monte Carlo EM algorithm," *Auton. Robots*, vol. 27, no. 2, pp. 123–130, 2009.

Ivo Grondman received the B.Sc. degree in applied mathematics and telematics from the University of Twente, Enschede, The Netherlands, in 2007, and the M.Sc. degree (with merit) in control systems from Imperial College London, London, U.K., in 2008. He is currently working toward the Ph.D. degree with the Delft Center for Systems and Control, Faculty of 3mE, Delft University of Technology, Delft, The Netherlands.

After graduation, he held a position with MathWorks, Cambridge, U.K., until 2009. His research interests include (approximate) dynamic programming and reinforcement learning, (optimal) control theory, neural networks, and computational intelligence methods.

Lucian Buşoniu received the M.Sc. degree (valedictorian) from the Technical University of Cluj-Napoca, Cluj-Napoca, Romania, in 2003, and the Ph.D. degree (*cum laude*) from the Delft University of Technology, Delft, The Netherlands, in 2009.

He is currently a Research Scientist with CNRS, Research Center for Automatic Control, University of Lorraine, Nancy, France. He is also with the Department of Automation, Technical University of Cluj-Napoca. He has authored a book as well as a number of journal, conference, and chapter publications on these topics. His research interests include reinforcement learning and dynamic programming with function approximation, planning-based methods for nonlinear stochastic control, multiagent learning, and, more generally, intelligent and learning techniques for control.

Dr. Buşoniu was the recipient of the 2009 Andrew P. Sage Award for the best paper in the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS.

Gabriel A. D. Lopes received the "Licenciatura" degree (Hons.) in aerospace engineering from Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal, in 1998, and the M.Sc. and Ph.D. degrees in electric engineering and computer science from the University of Michigan, Ann Arbor, in 2003 and 2007, respectively, with a grant from Fundação para a Ciência e Tecnologia, Portugal.

From 2005 to 2008, he was with the GRASP laboratory, University of Pennsylvania, Philadelphia. In 2008, he joined the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, as a Postdoctoral Fellow, where he has been an Assistant Professor since 2008. His research interests include geometric control, learning control, and discrete-event systems.

Dr. Lopes was a finalist for the Best Student Paper Award at the 2000 IEEE Conference on Decision and Control. While receiving the "Licenciatura" degree, he won a "Youth Researcher" grant for the project "Computational Dynamics of Articulated Human Motion."

Robert Babuška received the M.Sc. degree (*cum laude*) in control engineering from the Czech Technical University, Prague, Czech Republic, in 1990, and the Ph.D. degree (*cum laude*) from the Delft University of Technology, Delft, The Netherlands, in 1997.

He has had faculty appointments with the Technical Cybernetics Department, Czech Technical University Prague, and with the Electrical Engineering Faculty, Delft University of Technology. He is currently a Professor of intelligent control and robotics with the Delft Center for Systems and Control, Delft University of Technology. He has been working on applications of these techniques in the fields of robotics, mechatronics, and aerospace. His research interests include reinforcement learning, neural and fuzzy systems, identification and state-estimation, nonlinear model-based and adaptive control, and dynamic multiagent systems.