HAL
open science

Model-Free Reinforcement Learning with Continuous Action in Practice

Thomas Degris, Patrick M. Pilarski, Richard S. Sutton

To cite this version:

Thomas Degris, Patrick M. Pilarski, Richard S. Sutton. Model-Free Reinforcement Learning with Continuous Action in Practice. American Control Conference, Jun 2012, Montreal, Canada. hal-00764281

HAL Id: hal-00764281

https://inria.hal.science/hal-00764281v1

Submitted on 12 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Model-Free Reinforcement Learning with Continuous Action in Practice

Thomas Degris, Patrick M. Pilarski, Richard S. Sutton

Abstract— **Reinforcement learning methods are often considered as a potential solution to enable a robot to adapt to changes in real time to an unpredictable environment. However, with continuous action, only a few existing algorithms are practical for real-time learning. In such a setting, most effective methods have used a parameterized policy structure, often with a separate parameterized value function. The goal of this paper is to assess such actor–critic methods to form a fully specified practical algorithm. Our specific contributions include 1) developing the extension of existing incremental policy-gradient algorithms to use eligibility traces, 2) an empirical comparison of the resulting algorithms using continuous actions, 3) the evaluation of a gradient-scaling technique that can significantly improve performance. Finally, we apply our actor–critic algorithm to learn on a robotic platform with a fast sensorimotor cycle (10ms). Overall, these results constitute an important step towards practical real-time learning control with continuous action.**

I. PRACTICAL LEARNING IN ROBOTS

It is often desirable in robotics to allow the behaviour to change in real time in reaction to changes in the environment. A typical example is a robot vacuum cleaner that would adapt the parameters of its servo-motors depending on the floor texture to improve performance and save energy. Anticipating all the possible types of floor and adding sensors to determine when to change parameters would be difficult, if not impossible, and expensive. An alternative is to use online learning to track environmental changes and adapt to them (e.g., see [1]). If learning is in real time and continual, then the robot can constantly tune its internal parameters from its current experience while cleaning the room, continually adapting to changes in floor properties.

There have been surprisingly few reinforcement learning algorithms that have actually learned in real time on robotic platforms. For example, Kohl and Stone used reinforcement learning methods to improve the gait of a robot dog, but updated the policy only in an offline (though incremental) manner [2]. Similarly, the dynamic walking robot of Tedrake et al. used reinforcement learning to improve its policy, but made updates only at the end of each step cycle [3]. Peters and Schaal learned a swinging baseball task, but the policy was changed only at the end of one or more episodes [4]. A last example is Abbeel et al., who learned to fly an autonomous helicopter from offline data [5]. In addition to not being real-time learning, all these examples learn from a policy already well adapted to the problem: using pattern-generator-type policies [2][3] or imitation [4][5]. Depending on the problem, such policies may not be available.

P. M. Pilarski and R. S. Sutton are with the University of Alberta, Edmonton, Canada. T. Degris is with INRIA Bordeaux Sud-Ouest, France.

The clearest example of real-time reinforcement learning in robots we know of is that by Benbrahim et al. in 1993, who applied actor–critic methods with discrete action to a physically implemented "ball on a beam" balancing task; this system learned in real time at approximately 55ms per cycle [6]. The first work with real-time reinforcement learning on a conventional mobile robot seems to have been that by Bowling and Veloso in 2003 [7]. Their work also used discrete action and a time cycle of 100ms.

In practice, it can be difficult to apply reinforcement learning to real-time learning in robots. In robotics, actions are often continuous, whereas the overwhelming majority of work in reinforcement learning concerns discrete actions. Moreover, in order to be able to learn in real time, a reinforcement learning algorithm should ideally satisfy two key requirements. First, its per-time-step computational complexity should be linear in the number of learned weights in the policy parameterization. For example, the natural actor–critic algorithm [4] is not well suited to real-time use; it is often more sample efficient than a regular actor–critic algorithm, but its quadratic complexity is problematic when the number of weights is large or when the problem requires a fast update cycle. Second, the algorithm should be strictly incremental in that its per-time-step computational requirements do not increase with time [8].

In this paper, we bring together key contributions to form a fully specified practical algorithm. In particular, we build on the theoretical work of Bhatnagar et al. [9], extending their algorithms to continuous action, as pioneered by Williams [10], and to eligibility traces similar to Kimura et al. [11]. This paper is structured as follows. First, we introduce the algorithmic setting and theoretical framework. Within this framework, we describe a set of easy-to-implement algorithms, all with linear complexity. Then, we use an architecture based on tile coding [12] to examine the performance of the algorithms in two empirical studies. Finally, we demonstrate that our tile-coding architecture, with only limited a priori knowledge, is practical for real-time learning and control on a robot, and that, with adaptive exploration, the system is able to adapt its policy in a noisy non-stationary environment at a fast time scale (10ms).

II. THE POLICY GRADIENT FRAMEWORK

We consider a standard reinforcement-learning setting [12] except with a continuous action space $\mathcal{A}$. The state space $\mathcal{S}$ is assumed to be discrete just to simplify the presentation of the theory; in our experiments the state space is continuous. We use the policy-gradient framework in which a stochastic policy π is implicitly parameterized by a column vector of

weights $\mathbf{u} \in \mathbb{R}^N$, such that $\pi(a|s)$ denotes the probability density for taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. An objective function $J(\pi)$ maps policies to a scalar measure of performance. The principal idea in policy gradient methods is to improve the performance of a policy by updating its weight vector approximately proportionally to the gradient:

$$\mathbf{u}_{t+1} - \mathbf{u}_t \approx \alpha_u \nabla_{\mathbf{u}} J(\pi), \tag{1}$$

where $\alpha_u \in \mathbb{R}$ is a positive step-size parameter, and $\nabla_{\mathbf{u}} J(\pi) \in \mathbb{R}^N$ is the gradient of the objective function with respect to the policy weights $\mathbf{u}$.

A. Settings

Depending on the nature of the problem, two different settings can be considered when defining the objective function.

First is the average-reward setting, in which the interaction between the agent and its environment is continuing, without interruption or episodic termination. In this case, policies are evaluated according to the expectation of average reward per time step: $J(\pi) = \lim_{t\to\infty} \frac{1}{t} \mathrm{E}_\pi \left[r_1 + r_2 + \ldots + r_t\right]$.

Second is the starting-state setting, in which the agent seeks to maximize the total reward over an *episode* from a designated starting state s_0 up to a special terminal state. Policies are evaluated by the expectation of total discounted reward from s_0 until termination at time T (a random variable): $J(\pi) = \mathrm{E}_\pi \left[\sum_{t=1}^{T} \gamma^{t-1} r_t \middle| s_0\right]$ where $\gamma \in [0, 1]$ is known as the *discount-rate parameter*.

In either setting, we seek algorithms that approximate (1) on each time step. The rest of this section presents a theoretical forward-view analysis. The next section will convert the forward view to a backward view to produce online algorithms incorporating eligibility traces.

B. Forward View

The policy-gradient theorem [13] extends naturally to continuous actions as follows:

$$\nabla_{\mathbf{u}} J(\pi) = \sum_{s\in\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) Q^\pi(s,a) da, \tag{2}$$

where, for the average-reward setting, d^π is the limiting, stationary distribution of states under π, $d^\pi(s) = \lim_{t\to\infty} P(s_t = s|s_0, \pi)$, whereas, for the starting-state setting, $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0, \pi)$, the discounted state occupancy under π. The *action-value function* Q^π is defined by $Q^\pi(s,a) = \mathrm{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t - \bar{r}(\pi) | s_0 = s, a_0 = a\right]$ where, for the average-reward setting, $\gamma = 1$ and $\bar{r}(\pi) = J(\pi)$ is the average-reward of the policy, whereas, for the starting-state setting, $\bar{r}(\pi)$ is always 0. Finally, because $\int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) da = 0$, the policy-gradient equation can be generalized to include an arbitrary baseline function (e.g., see [9]), which we denote as $b : \mathcal{S} \to \mathbb{R}$, to yield

$$\nabla_{\mathbf{u}} J(\pi) = \sum_{s\in\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) \left[Q^\pi(s,a) - b(s)\right] da. \tag{3}$$

While using a baseline does not change the equality, it often decreases the variance of the gradient estimation. When a policy π is executed, the observed states are distributed according to $d^\pi(s)$, and the actions taken are according to π. In this case, (3) can be written as an expectation:

$$\nabla_{\mathbf{u}} J(\pi) = \mathrm{E}_\pi \left[\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \left(Q^\pi(s_t, a_t) - b(s_t)\right)\right], \tag{4}$$

where the expectation is over s and a sampled from their distributions, denoted $s \sim d^\pi(\cdot)$ and $a \sim \pi(\cdot|s)$. The vector $\frac{\nabla_{\mathbf{u}} \pi(a|s)}{\pi(a|s)}$ has been called the vector of *compatible features* [9], in other words, a gradient vector compatible with the features used to estimate the policy distribution. The next step is to write (4) as an expectation of returns [12]:

$$\nabla_{\mathbf{u}} J(\pi) = \mathrm{E}_\pi \left[\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \left(R_t - b(s_t)\right)\right], \tag{5}$$

where $s_t \sim d^\pi(\cdot)$, $a_t \sim \pi(\cdot|s_t)$, and the return $R_t = r_t - \bar{r}(\pi) + \gamma r_{t+1} - \bar{r}(\pi) + \gamma^2 r_{t+2} - \bar{r}(\pi) + \cdots$. The right-hand side of (5) now depends only on direct observations accessible while executing π.

III. ALGORITHMS

The return, R_t, is not directly available at time t because it depends on rewards that will be received on future time steps. As a first step toward solving this problem, the return can be approximated by the λ-*return* [12]:

$$R_t^\lambda = r_{t+1} - \bar{r}_t + \gamma(1-\lambda) v(s_{t+1}) + \gamma\lambda R_{t+1}^\lambda, \tag{6}$$

where $\lambda \in [0, 1]$, $\bar{r}_t$ is either an estimate at time t of the average reward, $\bar{r}(\pi)$, (for the average-reward setting) or uniformly 0 (for the starting-state setting), and finally $v(s_{t+1})$ is an estimate of the value of state s_{t+1} under π. From (5), we can now define a general forward-view algorithm for updating the policy weights:

$$\begin{aligned} \mathbf{u}_{t+1} - \mathbf{u}_t &= \alpha_u \left(R_t^\lambda - b(s_t)\right) \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \\ &= \alpha_u \delta_t^\lambda \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \end{aligned} \tag{7}$$

where s_t, a_t are the state and action at time t, and $\delta_t^\lambda = R_t^\lambda - b(s_t)$. While any function dependent only on the state could be used as a baseline, a natural choice is to use the estimate of the state value $v(s_t)$ for the policy, maintained by the critic of an actor–critic algorithm.

A. Backward View

Using the λ-return does not immediately solve the problem of the return depending on future rewards; note that, in (6), R_t^λ depends on its own future values, and thus it too depends on all the future rewards. Using eligibility traces and the backward view solves this problem. Using these, an incremental online update using only quantities available at each time step can closely approximate the forward view update (7). In this approach, the algorithm maintains a vector of traces $\mathbf{e}_t^{\mathbf{u}} \in \mathbb{R}^N$ of the eligibilities of each policy weight. We switch from the forward view to the backward view by

$$\mathbf{u}_{t+1} - \mathbf{u}_t = \alpha_u \delta_t^\lambda \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} = \alpha_u \delta_t \mathbf{e}_t^{\mathbf{u}} \tag{8}$$

where $\mathbf{e}^{\mathbf{u}}_t$ is the trace of the compatible features $\frac{\nabla_{\mathbf{u}}\pi(a_t|s_t)}{\pi(a_t|s_t)}$, updated by $\mathbf{e}^{\mathbf{u}}_t = \gamma\lambda\mathbf{e}^{\mathbf{u}}_{t-1} + \frac{\nabla_{\mathbf{u}}\pi(a_t|s_t)}{\pi(a_t|s_t)}$, and δ_t is the TD error defined as: $\delta_t = r_{t+1} - \overline{r}_t + \gamma v(s_{t+1}) - v(s_t)$.

B. Online Algorithms with Linear Complexity Per-time-step

We now present two algorithms with the following convention: for the average-reward setting, $\gamma = 1$ and $0 < \alpha_r < 1$ is the step-size for the estimate of the average reward, and for the starting-state setting, $\gamma \in [0, 1]$ and $\alpha_r = 0$.

From the backward view (8), it is straightforward to define an actor–critic algorithm with eligibility traces, denoted AC. AC is defined by:

Actor-critic Algorithm (denoted AC, and A when $\alpha_v = 0$)

Choose a according to $\pi(a|s)$
Take action a in s, observe s' and r
$\delta \leftarrow r - \overline{r} + \gamma\mathbf{v}^\mathsf{T}\mathbf{x}(s') - \mathbf{v}^\mathsf{T}\mathbf{x}(s)$
$\overline{r} \leftarrow \overline{r} + \alpha_r\delta$
$\mathbf{e}^{\mathbf{v}} \leftarrow \gamma\lambda\mathbf{e}^{\mathbf{v}} + \mathbf{x}(s)$
$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v\delta\mathbf{e}^{\mathbf{v}}$
$\mathbf{e}^{\mathbf{u}} \leftarrow \gamma\lambda\mathbf{e}^{\mathbf{u}} + \frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}$
$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u\delta\mathbf{e}^{\mathbf{u}}$

AC first updates its estimate of the average reward $\overline{r}$ (which can be considered as a trace of the immediate reward r). The state values are then estimated by the linear combination $v(s) = \mathbf{v}^\mathsf{T}\mathbf{x}(s)$, where $\mathbf{v}$ is a weight vector for the critic and $\mathbf{x}(s_t)$ is a feature vector corresponding to state s. First, the critic updates the weight $\mathbf{v}$ using the TD(λ) algorithm [14]; $\alpha_v > 0$ is its step-size parameter. Then, the policy weights $\mathbf{u}$ of the actor are updated based on the TD error δ and the eligibility trace $\mathbf{e}^{\mathbf{u}}$.

Algorithm AC is similar to that introduced by Kimura [11] (without the theoretical justification presented in this paper). AC without a critic ($\alpha_v = 0$), which we call A, is similar to REINFORCE and to OLPOMDP [15].

Our second algorithm, which we call the incremental natural actor–critic algorithm, or INAC, is new to this paper:

Incremental Natural Actor-critic Algorithm
(denoted INAC)

Choose a according to $\pi(a|s)$
Take action a in s, observe s' and r
$\delta \leftarrow r - \overline{r} + \gamma\mathbf{v}^\mathsf{T}\mathbf{x}(s') - \mathbf{v}^\mathsf{T}\mathbf{x}(s)$
$\overline{r} \leftarrow \overline{r} + \alpha_r\delta$
$\mathbf{e}^{\mathbf{v}} \leftarrow \gamma\lambda\mathbf{e}^{\mathbf{v}} + \mathbf{x}(s)$
$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v\delta\mathbf{e}^{\mathbf{v}}$
$\mathbf{e}^{\mathbf{u}} \leftarrow \gamma\lambda\mathbf{e}^{\mathbf{u}} + \frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}$
$\mathbf{w} \leftarrow \mathbf{w} - \alpha_v\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}^\mathsf{T}\mathbf{w} + \alpha_v\delta\mathbf{e}^{\mathbf{u}}$
$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u\mathbf{w}$

INAC extends an algorithm introduced by Bhatnagar et al. [9] to include eligibility traces. It uses the natural gradient $\tilde{\nabla}_{\mathbf{u}}J(\pi) = G(\mathbf{u})^{-1}\nabla_{\mathbf{u}}J(\pi)$, where $G(\mathbf{u})$ is the Fisher information matrix, $G(\mathbf{u}) = \sum_{s\in\mathcal{S}} d^\pi(s)\int_{\mathcal{A}}\pi(a|s)\frac{\nabla_{\mathbf{u}}\pi(a_t|s_t)}{\pi(a_t|s_t)}\frac{\nabla_{\mathbf{u}}\pi(a_t|s_t)}{\pi(a_t|s_t)}^\mathsf{T}$. As in the previous algorithm, the critic weights are updated with TD(λ). Then, the vector $\mathbf{w}$ is updated and used as an estimate of the natural gradient to update the actor weights.

Both algorithms AC and INAC converge for $\lambda = 0$, given some restrictions about the problem and the value of the parameters [9]. A convergence proof for $\lambda \neq 0$ is outside the scope of this paper, but Bhatnagar et al. [9] mention that their proof should extend to this case.

IV. POLICY DISTRIBUTION FOR CONTINUOUS ACTIONS

The algorithms mentioned above are independent of the structure of the policy distribution used in the policy. For discrete actions, the Gibbs distribution is often used. In this paper, for continuous actions, we define the policy such that actions are taken according to a normal distribution, as suggested by Williams [10], with a probability density function defined as $\mathcal{N}(s, a) = \frac{1}{\sqrt{2\pi\sigma^2(s)}}\exp\left(-\frac{(a-\mu(s))^2}{2\sigma^2(s)}\right)$ where $\mu(s)$ and $\sigma(s)$ are respectively the mean and standard deviation of the distribution $\pi(\cdot|s)$.

In our policy parameterization, the scalars $\mu(s) = {\mathbf{u}_\mu}^\mathsf{T}\mathbf{x}_\mu(s)$ and $\sigma(s) = \exp\left({\mathbf{u}_\sigma}^\mathsf{T}\mathbf{x}_\sigma(s)\right)$ are defined as linear combinations, where the parameters of the policy are $\mathbf{u} = \left({\mathbf{u}_\mu}^\mathsf{T}, {\mathbf{u}_\sigma}^\mathsf{T}\right)^\mathsf{T}$, and the features vector in state s is $\mathbf{x}_u(s) = \left(\mathbf{x}_\mu(s)^\mathsf{T}, \mathbf{x}_\sigma(s)^\mathsf{T}\right)^\mathsf{T}$.

The compatible features $\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}$ depend on the structure of the probability density of the policy. Given that our policy density is a normal distribution, the compatible features for the mean and the standard deviation are [10]:

$$\frac{\nabla_{\mathbf{u}_\mu}\pi(a|s)}{\pi(a|s)} = \frac{1}{\sigma(s)^2}\left(a - \mu(s)\right)\mathbf{x}_\mu(s) \quad (9)$$

$$\frac{\nabla_{\mathbf{u}_\sigma}\pi(a|s)}{\pi(a|s)} = \left(\frac{\left(a - \mu(s)\right)^2}{\sigma(s)^2} - 1\right)\mathbf{x}_\sigma(s) \quad (10)$$

where $\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)} = \left(\frac{\nabla_{\mathbf{u}_\mu}\pi(a|s)}{\pi(a|s)}^\mathsf{T}, \frac{\nabla_{\mathbf{u}_\sigma}\pi(a|s)}{\pi(a|s)}^\mathsf{T}\right)^\mathsf{T}$.

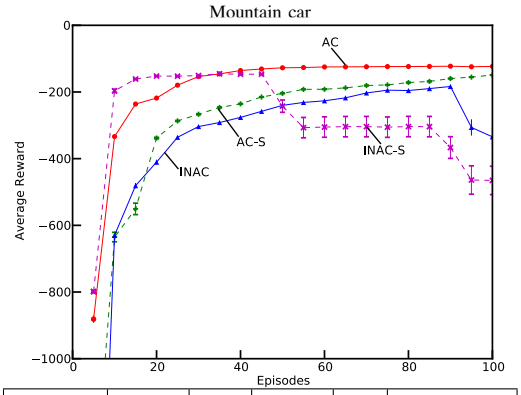
The compatible feature in (9), used to update the parameters $\mathbf{u}_\mu$ of the policy, has a $\frac{1}{\sigma(s)^2}$ factor: the smaller the standard deviation is, the larger the norm of $\frac{\nabla_{\mathbf{u}_\mu}\pi(a_t|s_t)}{\pi(a_t|s_t)}$ is, and vice-versa. We observed that such an effect can cause instability, particularly because $\lim_{\sigma\to 0}\frac{(a-\mu(s))}{\sigma(s)^2} = \infty$.

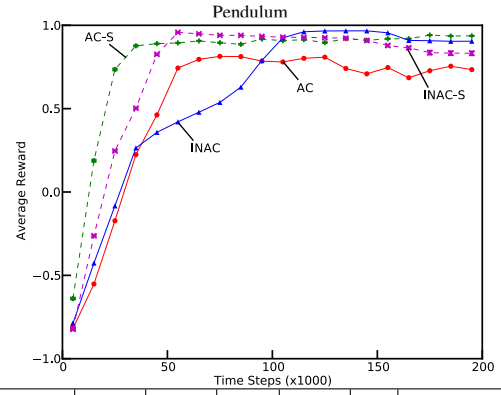
Williams [10] suggested the use of a step size of the form $\alpha_u\sigma^2$ for the gaussian distribution, scaling the gradient with respect to the variance of the distribution. We denote the actor–critic algorithm and the incremental natural actor–critic algorithm with this scaling gradient technique as, respectively, AC-S and INAC-S.

V. EMPIRICAL STUDY

We now present an empirical study evaluating how the ideas described above affect performance. This study includes using a critic, the natural gradient, eligibility traces and scaling the gradient estimate with respect to the variance.

In this paper, we used an architecture based on the *tile-coding* technique [12] to convert a continuous state to feature vectors. Tile coding takes an observation from the





	α_u	α_v	α_n	τ	Reward
A	.05	na	na	1	-3575 ± 42
	.005	na	na	32	-3297 ± 47
AC	.005	1.0	na	1	-340 ± 2
	.01	1.0	na	2	$-189\pm.5$
AC-S	.1	.5	na	1	-567 ± 40
	.01	1.0	na	2	-308 ± 1
INAC	.1	1.0	.0005	1	-435 ± 33
	.001	.5	.01	4	-386 ± 5
INAC-S	.01	1.0	.05	1	-286 ± 16
	.0005	1.0	.05	2	-518 ± 30

	α_r	α_u	α_v	α_n	τ	Reward
A	.5	.01	na	na	1	$-.53\pm.006$
	.05	.005	na	na	8	$.34\pm.004$
AC	.01	.005	.5	na	1	$.42\pm.003$
	.05	.01	.01	na	4	$.53\pm.001$
AC-S	.01	.05	.5	na	1	$.76\pm.001$
	.01	.05	.1	na	2	$.79\pm.001$
INAC	.005	.005	.5	.001	1	$.49\pm.003$
	.001	.001	1.0	.005	4	$.58\pm.003$
INAC-S	.005	.005	.5	.005	1	$.64\pm.002$
	.005	.005	.05	.005	8	$.70\pm.003$

Fig. 1. Top: *learning curve with standard error bars for the best parameters*. Bottom: *best parameter values and average reward per run with standard error. The best performing algorithms were* AC *and* AC-S *with eligibility traces. Gradient scaling often improved performance.* INAC *did not perform better than* AC.

environment as an input and expand it into a large sparse feature vector $\mathbf{x}$ which can then be linearly combined with a weight vector to represent non-linear functions. It is outside the scope of this paper to compare function approximation architectures, but tile coding has key advantages for real-time learning. First, tile coding is computationally efficient because computing the feature vector $\mathbf{x}$ does not depend on the size of $\mathbf{x}$ itself. Second, the norm of $\mathbf{x}$ is constant (i.e., the number of tilings). Finally, as we will show in the next section, tile coding is robust to noise.

The first problem is the mountain car problem [12], in which the goal is to drive an underpowered car to the top of a hill (starting-state setting). Actions are continuous and bounded in $[-1, 1]$. The reward at each time is -1 and the episode ends when the car reaches the top of the hill on the right or after 5,000 time steps. State variables consist of the position of the car (in $[-1.2, 0.6]$) and its velocity (in $[-.07, .07]$). The car was initialized with a position of -0.5 and a velocity of 0. We used $\gamma = 1$.

The second problem is a pendulum problem [16], in which the goal is to swing-up a pendulum in a vertical position (average-reward setting). The reward at each time is the cosine of the angle of the pendulum with respect to its fixed base. Actions—the torques applied to the base—are restricted to $a_t \in [-2, 2]$. State variables consist of the angle (in radians) and the angular velocity (in $[-78.54, 78.54]$). The pendulum was initialized and then reset every 1000 time steps in a horizontal position with an angular velocity of 0.

In both problem, we used ten 10×10 tilings over the joint space of both state variables, and a single constant feature. We used the same feature vector $\mathbf{x}(s)$ for the critic and for the mean and standard deviation of the actor ($\mathbf{x}(s)$, $\mathbf{x}_\mu(s)$, and $\mathbf{x}_\sigma(s)$). We performed 30 runs of a parameter sweep for the algorithms described above (A, AC, AC-S, INAC, and INAC-S) and for the parameters α_r, α_v and α_u with the following set of nine factors, $\{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, \ldots, .5, 1\}$, divided by the number of active features in $\mathbf{x}(s)$ (i.e., $10 + 1 = 11$). The parameter λ represents the rate at which the traces are decaying. Often, it is more intuitive to think about it in terms of the number of steps a trace will last. We use $\tau = \frac{1}{1-\lambda}$ to denote the decaying rate. We used the values $\{1, 2, 4, 8, 16, 32\}$ for τ in the parameter sweep. All the vectors and the average reward $\bar{r}$ were initialized to 0. In the starting-state setting, eligibility traces are reset to 0 at the beginning of each episode.

Figure 1 shows a summary of results. For readability each point is an average of the previous 10 episodes for mountain car and 10,000 time steps for the pendulum. We observe that, first, a critic drastically improved the performance. Second, the best performance for every algorithms was almost always with eligibility traces ($\tau > 1$), with only the exception of INAC-S on mountain car. Third, the scaling gradient technique often improved the performance, particularly on the pendulum problem. Last, the natural gradient used in INAC and INAC-S did not improve performance (compared to AC and AC-S), despite the additional step size α_n to set.

VI. REAL-TIME POWER CONSERVATION ON A MOBILE ROBOT

As a practical demonstration, we evaluated the effectiveness of AC-S on a robot. As mentioned in the introduction, an interesting continuous-action problem domain is power

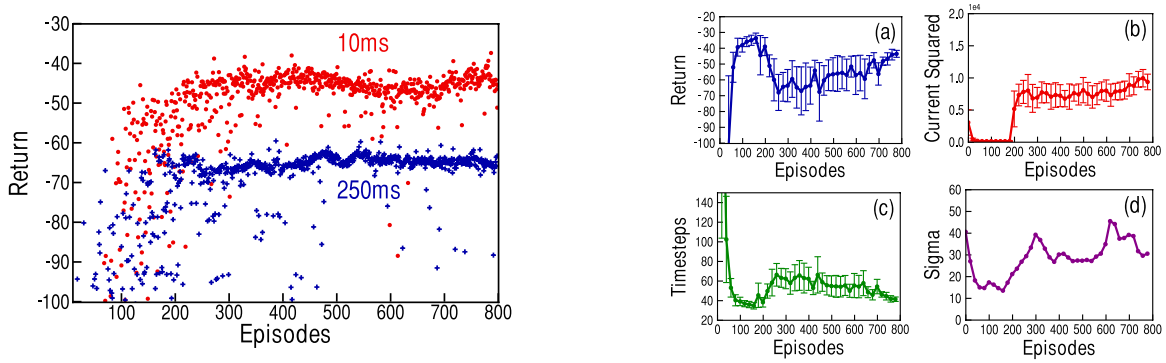


Fig. 2. Left: *Comparison of performance (total reward per episode) of the* AC-S *algorithm on the robot acceleration task at two different cycle times: 10ms and 250ms. Results are averaged over six independent runs at each cycle time. The average running time for the 800 episodes is 14.3 minutes.* Right: *Learning during the transition (episode 200) from suspended motion to moving on the ground in terms of (a) the return per episode, (b) the sum of the current squared per episode, and (c) the time steps until termination per episode; average of two runs. As shown in (d) for a single example run, the agent's learned σ value (i.e., level of exploration) increases after the transition to facilitate policy change. The average running time for the 800 episodes is 19.7 minutes.*

conservation on an autonomous mobile robot—for example, on a domestic service or remote search robot, where effective use of battery power can have a pronounced effect on the ability of the robot to carry out its task. During periods of acceleration, tradeoffs must be made between a robot's rate of acceleration to a desired set-point and the electrical current draw that this rate demands. Distinct environments may demand a different power/acceleration balance, making it essential to flexibly adapt motor control to new environments, even those unknown to designers.

The robot used in these experiments was a sensor-rich mobile robot with three omnidirectional wheels. The learning task was framed in an episodic fashion: each episode began with the robot in a stopped state and terminated when the measured speed $\dot{x}_t$ of one of the robot's three wheels reached or exceeded a target velocity level $\dot{x}_t \geq \dot{x}^*$. The agent was given control of a rotational motor command sent to the robot, allowing it to spin freely in either direction or remain stopped. The reward function for this problem was set as $r_t = -\left(\mathbf{1}_{\{\dot{x}_t < \dot{x}^*\}} + 0.5|i_t^2|\right)$, where $\mathbf{1}_{\{\dot{x}_t < \dot{x}^*\}}$ is an indicator function and i_t is the wheel current draw at time t. The task therefore was to balance two conflicting constraints in real-time: reaching $\dot{x}^*$ quickly (to avoid a negative reward on every time step) while at the same time minimizing the current used by its wheels. The velocity and current, $\dot{x}_t$ and i_t, constitute the available observations at each time step.

The feature vectors were created from the observations and the last action from joint tilings over their recent values. At each time t, we formed 10 tilings over $\langle \dot{x}_t, i_t, a_{t-1} \rangle$, 10 tilings over $\langle \dot{x}_{t-1}, i_{t-1}, a_{t-2} \rangle$ and so on up to $\langle \dot{x}_{t-4}, i_{t-4}, a_{t-5} \rangle$, for a total of 50 tilings overall. Each tiling had a resolution $10 \times 10 \times 10$, for a total of 50,000 features. Adding a single constant feature resulted in 50,001 binary features, exactly $m = 51$ of which were active at any given time. We used: $\alpha_v = 1.0/m, \alpha_u = 0.1/m, \alpha_r = \overline{r}_0 = 0, \gamma = 0.99$, and $\lambda = 0.7$. In the actor, only the update of the mean was multiplied by the variance. Weight vectors were initialized to 0; $\mathbf{u}_\sigma$ was initialized such that $\sigma = 40$, and bounded by $\sigma \geq 1$.

A. Experiment 1: Learning a Control Policy in Real Time

To explore AC-S learning performance at different degrees of real time operation, the robot acceleration task above was run with two different cycle times: 10ms and 250ms, where cycle time is the frequency at which actions are taken and the weight vectors updated. Each run lasted 800 episodes, and six runs were performed at each time cycle. This experiment demonstrated the ability of the AC-S algorithm to optimize a motor control policy when learning and action choice occured in real time.

At both cycle times, in less than 15 minutes (in average), AC-S learned in real-time viable control policies to deal with the constraints imposed by the reward function and the physical limits of each cycle time. We observed a statistically significant difference between asymptotic learning performance for the 10ms and 250ms cycle times (Figure 2, left). Learning at the faster 10ms cycle time led to higher long term reward; the robot used less current and demonstrated shorter acceleration periods, more effectively using battery power during acceleration.

B. Experiment 2: Online Control Policy Adaptation

One advantage of normal-distribution-based actor–critic methods is the role that σ plays in modulating exploration. This next experiment showed how AC-S was able to adapt its policy in response to an unexpected environmental change, while learning in real time (10ms cycle time). The experiential setup was similar to the previous experiment. However, at the beginning of the run, the robot was suspended so that its wheels did not touch the ground. After 200 episodes, the robot was moved so that it could spin in full contact with the ground. In this condition, the robot's mass and wheel friction played additional roles in making the trade-off between current draw and episode length different.

Figure 2 (right) shows how AC-S was able to successfully adapt the σ value used in action selection, modulating the degree of its exploration to deal with environmental

change. Results are shown here for the average of two independent runs, with each datapoint showing the binned average of twenty values. As shown in Figures 2a–c, AC-S converged on a policy for its initial environment (first 200 episodes). This was followed by a rapid decrease in reward per episode after the environmental change (episode 200), and then steady improvement as it learned a new policy for grounded movement (episodes 201–800). Upon encountering new environmental conditions—the transition from suspended movement to grounded movement—there was a significant increase in σ and a resulting change in behaviour. As AC-S learned to maximize return in this new situation (Figure 2a), it gradually adjusted the behaviour of the robot to decrease the number of time steps per episode (Figure 2c) while increasing the current per episode (Figure 2b). For one of the two runs (shown in Figure 2d), we found that σ increased a second time after encountering a superior policy choice near episode 600.

VII. DISCUSSION

The algorithms presented in this paper were all run on a standard laptop. Nevertheless, they still provided a response time more than sufficient to meet the real-time requirements of the problem, while taking decisions and learning. This is a key advantage of linear complexity actor–critic algorithms. Also, no model were required. No noise reduction filter was necessary to obtain the results presented in the paper.

A normal-distribution-based actor–critic allowed the robot to automatically adapt its policy in response to unexpected changes in the environment. For both robot application experiments, we found that the magnitude of σ decreased as AC-S converged to a new policy, but increased in response to new or altered environment dynamics. These results highlight the ability of actor–critic to adapt its exploration rate online in response to changes in the environment.

A problem with these methods is that it is necessary to determine the value of their step-size parameters. In practice, the following procedure can be used to set the parameters of AC and AC-S. First, for the average reward, α_r should be low (e.g., 0.001). Second, the rule of thumb of using 10% of the norm of the feature vector $\mathbf{x}$ can be used to set the critic step size α_v (while keeping $\alpha_u = 0$). Once the critic is stable, α_u can be progressively increased. Note that the reward function and the norm of the feature vector need be to bounded and normalized. An advantage of using tile coding is that it provides feature vectors with a constant norm.

Finally, robots often have more than one degree of freedom. While this work has been extended to an action space of two dimensions by Pilarski et al. [18], we do not think it can scale to high-dimension action space as such. However, we think that this work is a promising approach easily applicable to adapt time-dependent parameters in complex and/or non-stationary environments.

VIII. CONCLUSION

In this paper, we extended existing policy-gradient algorithms with eligibility traces and a gradient-skewing technique. We conducted an empirical study of existing and new algorithms on simulated and robotic tasks with continuous actions. The introduction of eligibility traces and gradient-scaling often improved performance, but the use of the natural gradient did not. Finally, this paper demonstrates that actor–critic algorithms with tile coding are practical to use for real-time learning with continuous action and state spaces to track unexpected environment changes.

IX. ACKNOWLEDGEMENTS

This work was supported by MPrime, the Alberta Innovates Centre for Machine Learning, the Glenrose Rehabilitation Hospital Fundation, Alberta Innovates—Technology Futures, NSERC, and Westgrid, a partner of Compute Canada. The authors gratefully acknowledge useful comments and discussion with Phillip Thomas and the reviewers.

REFERENCES

[1] Sutton, R. S., Koop, A., Silver, D. 2007. On the Role of Tracking in Stationary Environments. In *Proceedings of the 24th International Conference on Machine Learning*, 871–878.

[2] Kohl, N., Stone, P. 2004. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the International Conference on Robotics and Automation*, vol. 3, 2619–2624.

[3] Tedrake, R., Zhang, T., Seung, H. 2005. Stochastic Policy Gradient Reinforcement Learning on a Simple 3D Biped. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2849–2854.

[4] Peters, J., Schaal, S. 2008. Natural Actor-Critic. *Neurocomputing 71*(7–9):1180–1190.

[5] Abbeel, P., Coates, A., Ng, A. Y. 2010. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *International Journal of Robotics Research 29*(13):1608–1639.

[6] Benbrahim, H., Doleac, J. S., Franklin, J. A., Selfridge, O. G. 1992. Real-time Learning: A Ball on a Beam. *International Joint Conference on Neural Networks*.

[7] Bowling, M., Veloso, M. 2003. Simultaneous Adversarial Multi-robot Learning. *International Joint Conference on Artificial Intelligence*, 699-704.

[8] Sutton, R. S., Whitehead, S. D. 1993. Online Learning with Random Representations. *Proceedings of the Tenth International Conference on Machine Learning*, 314–321.

[9] Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., Lee, M. 2009. Natural Actor-Critic Algorithms. *Automatica 45*(11):2471–2482.

[10] Williams, R. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning 8*(3):229–256.

[11] Kimura, H., Yamashita, T., Kobayashi, S. 2002. Reinforcement Learning of Walking Behavior for a Four-Legged Robot. In *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 1, 411–416.

[12] Sutton, R. S., Barto, A. 1998. *Reinforcement Learning: an Introduction*. MIT press.

[13] Sutton, R. S., McAllester, D., Singh, S., Mansour, Y. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*, 1057–1063.

[14] Sutton, R. S. 1988. Learning to Predict by the Methods of Temporal Differences. *Machine Learning 3*(1):9–44.

[15] Baxter, J., Bartlett, P. 2002. Direct Gradient-Based Reinforcement Learning. In *Proceedings of the IEEE International Symposium on Circuits and Systems* vol. 3, 271–274.

[16] Doya, K. 2000. Reinforcement Learning in Continuous Time and Space. *Neural computation 12*(1):219–245.

[17] Tamei, T., Shibata, T. 2011. Fast Reinforcement Learning for Three-Dimensional Kinetic Human-Robot Cooperation with an EMG-to-Activation Model. *Advanced Robotics 25*(5):563–580.

[18] Pilarski, P. M., Dawson, M. R., Degris, T., Fahimi, F., Carey, J. P., Sutton R. S. 2011. Online Human Training of a Myoelectric Prosthesis Controller via Actor-Critic Reinforcement Learning. *Proceeding of the IEEE International Conference on Rehabilitation Robotics*, 134–140.