Learning Continuous-Action Control Policies

Jason Pazis and Michail G. Lagoudakis

***Abstract*— Reinforcement Learning for control in stochastic processes has received significant attention in the last few years. Several data-efficient methods, even for continuous state spaces, have been proposed, however most of them assume a small and discrete action space. While continuous action spaces are quite common in real-world problems, the most common approach still employed in practice is coarse discretization of the action space. This paper presents a novel, computationally-efficient method, called Adaptive Action Modification, for realizing continuous-action policies, using binary decisions corresponding to adaptive increment or decrement changes in the values of the continuous action variables. The proposed approach essentially approximates any continuous action space to arbitrary resolution and can be combined with any discrete-action reinforcement learning algorithm for learning continuous-action policies. Our approach is coupled with three well-known reinforcement learning algorithms (Q-Learning, Fitted Q-Iteration, and Least-Squares Policy Iteration) and its use and properties are thoroughly investigated and demonstrated on the continuous state-action Inverted Pendulum and Bicycle Balancing and Riding domains.**

I. Introduction

EVENTS around us are continuous. The torque applied by a muscle, the speed of a moving vehicle, the current supplied on a power line, are all examples of quantities typically described by continuous variables. Computers, on the other hand, are digital and manipulate discrete data using discrete structures and methods. Variations of this discrete/continuous "discrepancy" can be found in various disciplines and have been addressed with varying degrees of success. Some familiar examples are the following:

- given an analog voice signal, what is the best method of transmitting it over a digital communication channel?
- given an audio recording, how can we compress and store it with the minimum possible loss of quality?
- given a picture, how can we utilize the similarities between pixels to achieve a more compact representation?
- given a video recording, how can the similarity between successive frames be utilized to achieve compression?

The answer to these and other related questions, exploits and utilizes some form of structural locality (temporal and/or spatial). Locality is a recurring theme in many scientific disciplines and harnessing its properties can offer some very important advantages.

Focusing our attention to research in Reinforcement Learning (RL), we can easily notice that the majority of known algorithms are able to learn policies that make decisions over a small set of discrete actions, but run into trouble when the number of actions grows beyond a certain point, not to mention cases where actions are continuous. A number of efforts have attempted to find a way around this problem; some have met more success than others, yet none can be characterized as a comprehensive solution. Our motivation for this work stems from the belief that exploitation of temporal locality in RL domains can offer viable generic solutions with wide applicability to real-world problems.

This paper addresses the problem of learning control policies in stochastic domains where actions (and states) are continuous. Efficiency and scalability are carefully considered during every step of the process to ensure feasibility in real-world applications. Our contribution is a highly-efficient, generic algorithm which allows the majority of known RL algorithms to learn and execute effective continuous-action control policies in domains with temporal action locality. The proposed approach, called Adaptive Action Modification, essentially approximates any continuous action space to arbitrary resolution and requires only binary decisions on behalf of the learning algorithm to adaptively modify the values of the continuous action variables. The viability of the proposed approach is demonstrated in conjunction with three well-known RL algorithms (Q-Learning, Fitted Q-Iteration, and Least-Squares Policy Iteration) on two well-known RL domains. To our knowledge this is the first time the barrier of continuous action spaces is overcome in such a way.

The remainder of the paper is organized as follows. Section II provides the necessary background material and Section III discusses the issue of continuous actions. The new algorithm is presented in Section IV and its benefits and properties are experimentally demonstrated in Section V. The paper concludes with a detailed discussion in Section VI.

II. Background

A. Markov Decision Processes

A *Markov Decision Process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is the action space of the process, P is a Markovian transition model $\big(P(s, a, s')$ denotes the probability of a transition to state s' when taking action a in state $s\big)$, R is a reward function $\big(R(s, a)$ is the expected reward for taking action a in state $s\big)$, $\gamma \in (0, 1]$ is the discount factor for future rewards, and D is the initial state distribution. A *deterministic policy* π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice in state s. The value $Q_\pi(s)$ of a state-action pair (s, a) under a policy π is defined as the expected, total, discounted reward when the process begins in state s, action a is taken at the first step, and all

Jason Pazis and Michail G. Lagoudakis are with the Department of Electronic and Computer Engineering, Technical University of Crete, Chania, Crete, Greece, (email: {jpazis, lagoudakis}@intelligence.tuc.gr).

This work was partially supported by the Marie Curie International Reintegration Grant MCIRG-CT-2006-044980 awarded to Michail G. Lagoudakis within the 6th European Framework Programme.

Fitted Q-Iteration $(\mathcal{D},\ \gamma,\ N)$
$k \leftarrow 0$, $Q^k \leftarrow \mathbf{0}$
do
$TS \leftarrow \emptyset$
for each (s, a, r, s') in $\mathcal{D}$
$TS \leftarrow TS \cup \left\{\left((s,a),(r+\gamma\ \max_{a' \in \mathcal{A}} Q^k(s',a'))\right)\right\}$
$Q^{k+1} \leftarrow$ use regression on TS
$k \leftarrow k+1$
while $(k < N)$
return Q^N

Fig. 1. The Fitted Q-Iteration algorithm.

LSPI $(\mathcal{D},\ \phi,\ \gamma,\ \epsilon)$
$w' \leftarrow \mathbf{0}$
repeat
$w \leftarrow w'$, $\mathbf{A} \leftarrow \mathbf{0}$, $b \leftarrow \mathbf{0}$
for each (s, a, r, s') in $\mathcal{D}$
$a' = \arg\max_{a'' \in \mathcal{A}} w^\top \phi(s', a'')$
$\mathbf{A} \leftarrow \mathbf{A} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s',a')\Big)^\top$
$b \leftarrow b + \phi(s,a)r$
$w' \leftarrow \mathbf{A}^{-1}b$
until $(\|w - w'\| < \epsilon)$
return w

Fig. 2. The LSPI algorithm.

decisions thereafter are made according to policy π:

$$Q_\pi(s,a) = E_{a_t \sim \pi; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, a_t\big) \Big| s_0 = s, a_0 = a\right] .$$

The goal of the decision maker is to find an optimal policy π^* for choosing actions, which maximizes the expected, total, discounted reward from the initial state distribution D:

$$\pi^* = \arg\max_\pi E_{s \sim D}\left[Q_\pi\big(s, \pi(s)\big)\right] .$$

If the value function Q_{π^*} is known, a greedy policy, which simply selects actions that maximize Q_{π^*} in each state s, is an optimal policy. It is known that for every MDP, there exists at least one optimal deterministic policy. Value iteration, policy iteration, and linear programming are well-known methods for deriving an optimal policy from the MDP model.

B. Reinforcement Learning

In reinforcement learning, a learner interacts with a process modeled as an MDP and typically observes the state of the process and the immediate reward at every step, however P and R are not accessible. The goal is to gradually learn an optimal policy using the experience collected through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, thus experience comes in the form of (s, a, r, s') samples. Several algorithms have been proposed for learning good or even optimal policies from samples [1].

Q-Learning [2] is probably the most popular reinforcement learning algorithm. It maintains an estimate Q of the optimal value function Q_{π^*}, which is updated for each new sample (s, a, r, s') using the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma\ \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)\right)$$

Q-Learning is a stochastic approximation algorithm (with a step-size α), which imposes limited computational demands at every step, but comes with the drawback that it usually takes a large number of steps to converge. The learned policy is the greedy policy over the learned value function Q.

Fitted Q-Iteration [3] is a batch-training version of the Q-Learning algorithm. It uses an iterative scheme to approximate the optimal value function, whereby an improved value function Q is learned at each iteration by fitting a function approximator to a set of training examples generated using a set of samples from the process and the Q-Learning update rule. Any function approximation architecture and the corresponding supervised learning algorithm could be used in the iteration. For certain classes of function approximation architectures there exists proof of convergence [3], [4]. In practice, Fitted Q-Iteration (summarized in Figure 1) produces very good policies within a moderate number of iterations.

Least-Squares Policy Iteration (LSPI) [5] is a reinforcement learning algorithm based on the approximate policy iteration framework, whereby at each iteration an improved policy is produced as the greedy policy over an approximation of the value function of the previous policy. LSPI uses linear approximation architectures consisting of a weighted sum of a set of basis functions ϕ for representing value functions. The value function of each policy is learned by solving a linear system formed using a set of samples from the process and the fixed-point property of the value function under the Bellman equation. Although, there is no guarantee that improvement will be monotonic, it is guaranteed that the iteration will not diverge, and indeed in most cases it converges in a few iterations, otherwise it oscillates within a small region. LSPI is summarized in Figure 2.

C. Adaptive Delta Modulation

The approach proposed in this paper relies on the concept of Adaptive Delta Modulation, which is a special case of Differential Pulse-Code Modulation (DPCM) [6]. Very often a data stream contains samples that are highly correlated. In such cases, an efficient transmission method is to code the difference of each sample from the previous one. Using a high enough sample rate, the difference between successive samples can be encoded into 1 bit [6]. The quantizer has two levels, $\pm\Delta$, therefore each sample differs by $\pm\Delta$ from the previous one. At the receiver the stream can be recovered as

$$\hat{X}_t = \hat{X}_{t-1} + e_t\Delta ,$$

where $\hat{X}_t$ is the (estimated) original data sample at time step t and e_t is the 1-bit sign of the difference received at time step t. The magnitude of Δ is a very important design factor. Large values for Δ allow the system to follow fast changes of the input, but cause too much quantization noise, whereas

when Δ is too small it cannot follow a rapid changing input. A very simple, yet effective, solution to the above problems, is to change the magnitude of Δ adaptively, depending on the input. The simplest rule for adjusting Δ is

$$\Delta_t = \Delta_{t-1} K^{e_t e_{t-1}} ,$$

where K is a real-valued constant greater than one.

III. Continuous Control Actions

The most common approach adopted in learning continuous-action controllers is a coarse discretization of the action space. For instance, in the well-known Inverted Pendulum domain (cf. Section V) three actions are usually allowed: left force (-50 Newtons), right force ($+50$ Newtons), or no force (0 Newtons), whereas ideally one would like to be able to choose any action in the range $[-50N, 50N]$. In reality, no physical system can switch instantly from exerting a force of -50 Newtons to exerting a force of $+50$ Newtons; this fact poses a reasonable question as to whether all these –successful in simulation– discrete-action controllers would be as successful in the real world. Even if physical limitations are overcome, the resulting controllers will be very abrupt on their action choices, resulting in rather jerky control. While succeeding in keeping a plant within some desired margin is deemed adequate for research purposes, in a commercial application it is equally important to also ensure that energy consumption stays as low as possible, mechanical stresses are minimized, noise induced to the power lines is kept within acceptable limits, motion is smooth, etc. There is also a subtlety inherent to discrete-action policies that significantly complicates things. One would hope that smooth changes in state should incur smooth changes in action. Yet with discrete actions that cannot be the case; for states close to the decision boundaries, tiny changes in state can result in completely different actions.

Most existing approaches in the literature trying to tackle the continuous-action learning problem rely on specialized cases of neural networks [7]–[10]. While for some applications neural networks are suitable, there are cases where they are not easily applicable. The main concern is the computational resources required. Many classical low-cost control applications run either on simple analog circuits or on humble 8-bit microcontrollers. To be competitive at a commercial level, successful reinforcement learning methods would have to be able to run on comparable hardware. The exponent needed to evaluate each neuron, makes them too expensive for such applications. Other proposals and implementations [11], [12] attempt to find an "optimal" finite subset of discrete actions or a set of actions that can slowly "shift" over the state space when the state changes slowly. One common problem with these approaches, is their reliance on specific-for-the-task learning methods, which make inefficient use of samples and require online, on-policy learning. This fact makes them difficult to use on a real system, where samples are expensive. Another concern with these approaches and others involving Markov Chain Monte Carlo (MCMC) sampling [13] or one step lookahead [14] is that they are quite computationally demanding, even more so than the neural-network based implementations. They also scale badly when the desired resolution or the number of controlled variables increases. M. Riedmiller [15] describes the Dynamic Output Element (DOE) approach, which generates continuous control signals within a self-learning neural control architecture by integrating discrete actions over time. Although it takes advantage of temporal locality, it requires a moderate-sized set of discrete actions in order to achieve good resolution and speed, due to the fact that the input to the integrator is non-adaptive. The idea presented in this paper could be considered a special case of adaptive DOE elements.

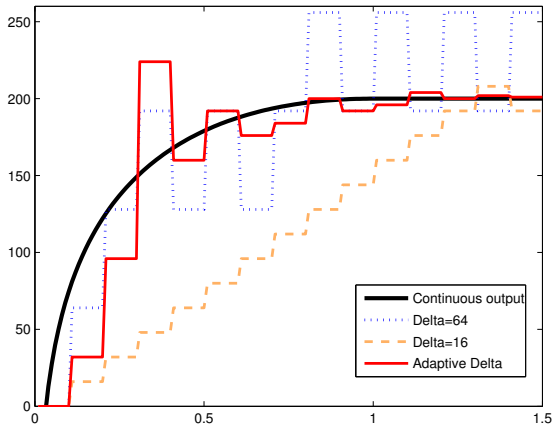

Fig. 3. Desired action output and approximation with various Δ's.

IV. Adaptive Action Modification

A. The Proposed Algorithm

A policy for an MDP typically selects the action to be performed next. However, when dealing with continuous action spaces over a range of values, successive decisions exhibit a great deal of temporal locality. So, instead of making a decision about what particular action to perform, one can make a decision on how to modify the current action. In its simplest form this could be a decision whether to increase or decrease the current action value by a certain amount. The problem now is to find the size of the modification step (Δ) in the action space. If the step is too large, the controller can respond to quick changes, but the resolution is effectively limited; whereas, if the step is too small, the resolution is increased, but the controller is unable to respond to fast changes. There is a simple way to get the best of both worlds, which has long been used in telecommunication systems: change the size of the step adaptively. If for two successive time steps the sign of the decision remains the same (with increase taken as positive and decrease as negative), the step-size Δ is increased by a real-valued factor $K > 1$ to $K \times \Delta$, whereas when the sign of two successive decisions

AAM $(s, \pi, e_{t-1}, A_{t-1}, A_{res}, A_{\min}, A_{\max}, \Delta_{t-1}, \Delta_{\min}, \Delta_{\max})$

// s : The current state of the process
// π : A policy making binary decisions, $+1$ or -1
// e_{t-1} : The last decision of policy π
// A_{t-1} : The last value of the continuous action A
// A_{res} : The desired resolution over the continuous-action space
// $A_{\min}$: Minimum allowed value of A
// $A_{\max}$: Maximum allowed value of A
// Δ_{t-1} : The last value of the step size Δ
// $\Delta_{\min}$: Minimum allowed value of Δ (default: 1)
// $\Delta_{\max}$: Maximum allowed value of Δ (default: A_{res})

$e_t \leftarrow \pi(s, e_{t-1}, A_{t-1}, \Delta_{t-1})$ // make a binary decision
$\Delta_t \leftarrow \Delta_{t-1} K^{e_t e_{t-1}}$ // update Δ

if $(\Delta_t > \Delta_{\max})$ // make sure Δ stays within acceptable limits
  $\Delta_t \leftarrow \Delta_{\max}$
else if $(\Delta_t < \Delta_{\min})$
  $\Delta_t \leftarrow \Delta_{\min}$

// update A according to the new Δ_t and the decision e_t
$A_t \leftarrow A_{t-1} + e_t \Delta_t \left(\frac{A_{\max} - A_{\min}}{A_{res} - 1} \right)$

if $(A_t > A_{\max})$ // make sure A stays within acceptable limits
  $A_t \leftarrow A_{\max}$
else if $(A_t < A_{\min})$
  $A_t \leftarrow A_{\min}$

return A_t, Δ_t, e_t

Fig. 4. The Adaptive Action Modification algorithm.

is different, the size of the step is decreased to $K^{-1} \times \Delta$. Figure 3 illustrates this point.

Given these observations, it is now straightforward to design an algorithm that takes advantage of temporal locality in actions. The resulting algorithm, called Adaptive Action Modification (AAM), uses a discrete, binary-decision policy that chooses whether A, the continuous action, will be increased or decreased at each step. The step-size Δ is adjusted appropriately, according to a scaling factor K, and the value of A is updated. The whole process is repeated at each time step. Figure 4 summarizes the proposed algorithm.

B. Efficiency, Scalability, Effectiveness

The motivation behind the proposed algorithm was computational efficiency and ease of implementation on low cost microcontrollers. Apart from the evaluation of the binary-decision policy, whose cost is directly dependent on the representation used, the remainder of the algorithm is amenable to a low-cost implementation. When K is a power of 2, the updates of the step size Δ and the continuous action A can be computed with just a few additions, subtractions, shift operations, and conditional statements. For example, for $K = 2$ the updates could be computed as ($\ll$ is the left shift operator):

if $((e_t + e_{t-1}) == 0)$
  $\Delta_t \leftarrow \Delta_{t-1} - 1$
else
  $\Delta_t \leftarrow \Delta_{t-1} + 1$

if $(e_t == 1)$
  $A_t \leftarrow A_{t-1} + ((A_{\max} - A_{\min})/(A_{res} - 1)) \ll \Delta_t)$
else
  $A_t \leftarrow A_{t-1} - ((A_{\max} - A_{\min})/(A_{res} - 1)) \ll \Delta_t)$

Note that $((A_{\max} - A_{\min})/(A_{res} - 1))$ is a constant and can be precomputed and replaced in the expression. To our knowledge, this is the least computationally-intensive method for generating continuous actions.

If the controller has to make simultaneous decisions for n continuous action variables, a policy choosing among 2^n discrete joint action choices is required (one binary modification decision for each one of the n action variables), which is admittedly expensive. Nevertheless, a discrete controller over a discretized action space offering m choices per action variable, would need a policy choosing among m^n joint actions for n action variables. Clearly, m cannot be less than 2 and, in fact, m is usually much larger than 2, therefore the requirements of our controller on the policy do not exceed those of a minimalist discrete controller.

One could argue that the proposed algorithm does not truly offer continuous actions the way other approaches do. Partitioning an action range to a resolution of 2^8 or even 2^{16} does provide a large number of actions over that range, but actions are still discrete, not continuous. While in principle this statement is true, one has to take into consideration how controllers are implemented on real hardware. Analog-to-Digital and Digital-to-Analog converters have finite precision, which for the majority of cases on modern microcontrollers is at most 2^{16}. Also, the control frequency is limited by the time it takes to complete one iteration of the control algorithm on real (limited) hardware. Therefore, precision over a continuous-action range is practically limited and our approach allows for approximating this continuous-action range to arbitrary precision with no real increase in computational cost (although when there is only limited temporal locality of actions, we may have to increase the control frequency to allow for quick action adaptation, thereby indirectly increasing the computations per second).

C. Integration with Reinforcement Learning Algorithms

Most existing reinforcement learning algorithms can be used in conjunction with the proposed method to learn policies over continuous action spaces. There are only two requirements on the learning algorithm of choice: (a) it must be able to handle continuous state spaces, and (b) it must be able to produce a binary decision for each controlled variable. The first requirement is dictated by the fact that the state space may have to be enhanced by additional continuous state variables. The original control problem comes with a state space $\mathcal{S}$. In order to preserve the Markov property on the process when using the proposed algorithm, the state must include the original state variables, as well as the latest values of A, Δ, and e. These are internal, local variables and their values could be made available to the learner to enhance its state description; alternatively, we can choose to hide any of them and treat the resulting process as a Partially Observable MDP (POMDP). Our empirical evidence suggests that inclusion of the latest value of A alone

is a good trade-off for achieving a good level of performance without increasing significantly the dimension of the state space. The second requirement is dictated by the need of a separate modification decision for each action variable.

The fundamental principle behind the proposed technique relies on taking advantage of the temporal locality of continuous actions. Nevertheless, in domains where the action must change quickly, we may have to increase the control frequency to accommodate for quicker changes. This modification has the potentially negative consequence that, during learning, more samples will be required to cover the same amount of real time. Due to the fact that good actions and delayed rewards become separated by more steps, the ill effects of the Temporal Credit Assignment problem in reinforcement learning are amplified. This problem may be alleviated by using more direct or shaping reward functions.

V. Experimental Results

We have integrated Adaptive Action Modification with three well-known reinforcement learning algorithms: Least-Squares Policy Iteration (LSPI), Q-Learning, and Fitted Q-Iteration. Unless otherwise mentioned below, the names of these three algorithms will refer to their enhanced versions. The value of K in all experiments was set to 2 for the sake of simplicity, although it could be optimized for better performance. For all three algorithms, we used the same linear approximation architecture (the same set of basis functions) for representing the value function in each domain, to factor out differences due to representational capacity. Therefore, observed differences in performance among the three algorithms are solely due to their ability to learn an appropriate set of weights. Least-Squares Regression was used for fitting within the Fitted Q-Iteration algorithm.

A. Inverted Pendulum

The inverted pendulum problem [16] requires balancing a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. The 2-dimensional continuous state space includes the vertical angle θ and the angular velocity $\dot{\theta}$. Three discrete actions are typically used: left force (-50 Newtons), right force ($+50$ Newtons), or no force (0 Newtons). However, in our experiments the action space consists of the entire range of forces $[-50N, 50N]$ approximated to a resolution of 2^8 equally spaced actions. The state was augmented to include the most recent value of the action F, therefore it becomes a 3-dimensional vector $(\theta, \dot{\theta}, F)$. All actions are noisy; uniform noise in $[-10N, 10N]$ is added to the chosen action. The transitions are governed by the nonlinear dynamics of the system and depend on the current state and the current (noisy) control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)} \quad ,$$

where g is the gravity constant ($g = 9.8m/s^2$), m is the mass of the pendulum ($m = 2.0$ kg), M is the mass of the cart ($M = 8.0$ kg), l is the length of the pendulum ($l = 0.5$ m), and $\alpha = 1/(m + M)$. The simulation step is set to 0.02 seconds and the control input is given at the beginning of each time step. A reward of $-|2\theta/\pi|$ is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value. An angle greater than $\pi/2$ signals the end of the episode and a reward of -1. The discount factor of the process is 0.95.

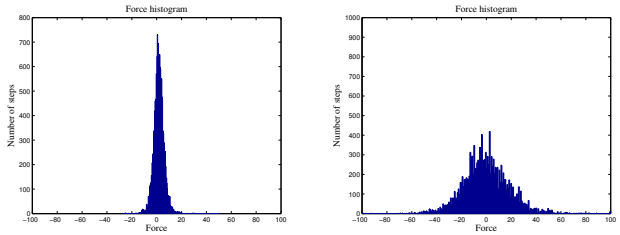

Fig. 5. Inverted pendulum: Force histograms of policies learned by LSPI. Left: $50N$ case, $10N$ noise. Right: $100N$ case, $50N$ noise.

The approximation architecture for representing the value function in this problem consists of a total of 56 basis functions. In particular, for each one of the two choices of the modification policy (increase/decrease), there is a separate set of 28 basis functions, including a constant term and 27 radial basis functions arranged in a $3 \times 3 \times 3$ grid over the 3-dimensional normalized state space:

$$\phi = \Big(1 \quad , \quad e^{-\frac{(\theta-\theta_1)^2/(\pi/4)^2 + (\dot{\theta}-\dot{\theta}_1)^2/2.5^2 + (F-F_1)^2/50^2}{2\sigma^2}} \quad , \quad e^{-\frac{(\theta-\theta_2)^2/(\pi/4)^2 + (\dot{\theta}-\dot{\theta}_2)^2/2.5^2 + (F-F_2)^2/50^2}{2\sigma^2}} \quad , \quad \cdots \quad , \quad e^{-\frac{(\theta-\theta_3)^2/(\pi/4)^2 + (\dot{\theta}-\dot{\theta}_3)^2/2.5^2 + (F-F_3)^2/50^2}{2\sigma^2}}\Big)^\top ,$$

where the θ_i's are $\{-\pi/4,\ 0,\ +\pi/4\}$, the $\dot{\theta}_i$'s are $\{-2.5,\ 0,\ +2.5\}$, the F_i's are $\{-50,\ 0,\ +50\}$, and $\sigma^2 = 1$.

1) LSPI: The performance of the learned controllers with LSPI was excellent; the pendulum stayed at the upright position ($\theta \approx 0$) for the duration of testing (15,000 steps or 5 minutes). Angular velocity and acceleration were also concentrated close to zero, with no sudden spikes. The Δ's used by the controller were mainly from the low end of the spectrum, which resulted in small steps in the action space, taking advantage of the available resolution. Figure 5 (left) is a histogram of the applied force over the $[-50N, 50N]$ range for a sample test run. The majority of applied forces is concentrated around zero. The average mean force magnitude over 100 learned policies on $[-50N, 50N]$ was $4.37N$. It is important to note that the algorithm yields such small mean force magnitudes, since it does not need very large forces to counteract the $[-10N, 10N]$ uniform noise.

We further increased the available action (force) range to $[-100N, 100N]$ without changing the resolution from 2^8. The performance of the learned controller and the results for the angle, velocity and acceleration were practically the same. The Δ's used, stayed at even lower values, due to the fact that each step in this action space is larger. The average mean force magnitude over 100 learned policies on $[-100N, 100N]$ was $4.52N$, only marginally higher than the one in the $50N$ case.

2) Performance under increasing noise: In the above experiments, the Δ's, as well as the force magnitudes, are concentrated at very low values. Although this is generally a good thing, a question arises as to whether the learned controllers can use the rest of the range when the situation requires it. To test this hypothesis, we fixed a learned policy and progressively increased the noise level, up to a point, where the noise had the same magnitude as the controlling action. It was observed that the higher the noise magnitude, the larger the Δ's and the force magnitudes used; nevertheless, the learned controller did not loose control of the pendulum, instead it kept it balanced at all times. Figure 5 (right) shows the force histogram for the $[-100N, 100N]$ controller, when noise is increased to $[-50N, 50N]$.

3) Comparison with the discrete three-action controller: To assess the benefit of Adaptive Action Modification over the coarse discretization approach, we compared the performance of the learned continuous-action controllers to the performance of the learned discrete three-action controllers. Both the $(-50N, 0N, 50N)$ and the $(-100N, 0N, 100N)$ learned controllers succeed in keeping the pendulum balanced with the angle very close to zero. However, significant differences are observed as far as angular velocity, acceleration, and force magnitude are concerned. Angular velocity and acceleration reach much higher values with the discrete controllers than with the continuous-action controllers, especially in the $100N$ case. The average mean force magnitude over 100 learned discrete controllers was $8.30N$ for the $50N$ case and $50.60N$ for the $100N$ case. Comparing these numbers to the corresponding mean force magnitudes of the continuous-action controllers ($4.37N$ and $4.52N$), the benefit of using a continuous-action controller becomes obvious. On a real system, the force used is directly related to power consumption and mechanical stresses.

4) Q-Learning/ER: The experiments above were repeated using the Q-Learning algorithm enhanced with the experience replay (ER) technique for learning the modification policy. Experience replay allows the reuse of samples for updating the value function multiple times. The controllers learned with Q-Learning are comparable to those of LSPI, although Q-Learning frequently produced policies that balanced the pendulum at an angle slightly different than 0.

5) Fitted Q-Iteration: The experiments above were repeated using Fitted Q-Iteration for learning the modification policy. A rather unexpected behavior was observed during learning. With increased number of iterations, the policies did not necessarily improve. In fact, although policies produced at the 20th iteration were very successful, if the algorithm was allowed to run up to 100 iterations with the same set of samples, almost none of the policies were still able to balance the pendulum. Since the 20th iteration seemed to produce consistently good results, training for all experiments was cut off at that point. Again, the resulting controllers with Fitted Q-Iteration were comparable to those produced with LSPI and Q-Learning, although they exhibited worse mean force magnitude. The average mean force magnitude over 100 learned controllers was $5.65N$ for the $[-50N, 50N]$ range and $6.24N$ for the $[-100N, 100N]$ range.

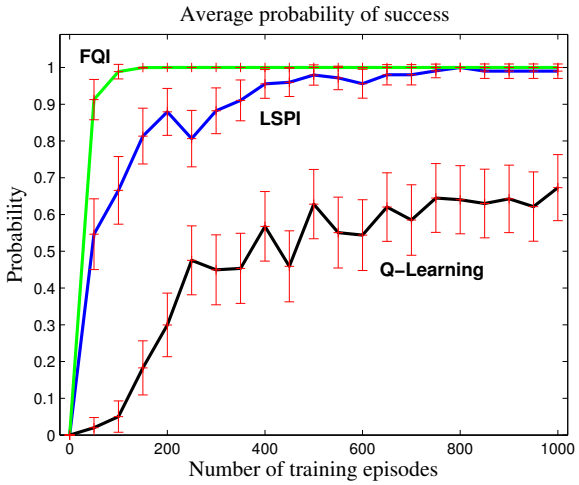

Fig. 6. Inverted pendulum : Average probability of success ($100N$ case).

6) Learning Curves: A number of systematic experiments were conducted to assess the effectiveness of learning under the proposed continuous-action control scheme. Training samples were collected in advance from "random episodes", that is, starting the pendulum in a randomly perturbed state very close to the equilibrium state $(0, 0, 0)$ and following a policy that selected binary increase/decrease actions uniformly at random. The average length of such episodes was about 36 steps for the $100N$ and 42 steps for the $50N$ case, thus each run contributed about 36 or 42 samples to the set respectively. Figure 6 shows the performance of the learned policies for the $100N$ case, as a function of the number of training episodes. For each batch of training episodes, the learned policy was evaluated 100 times to estimate accurately the average number of balancing steps. This experiment was repeated 100 times for the entire horizontal axis to obtain average results and the 95% confidence intervals over different sample sets. Each episode was allowed to run for a maximum of $15,000$ steps corresponding to 5 minutes of continuous balancing in real-time. A run that balanced for this period of time was considered to be successful. Fitted Q-Iteration was by far the fastest learner yielding fully successful controllers with only a few hundred episodes. LSPI's success was good as well and, as mentioned before, the performance of successful policies was better. Q-Learning was far less successful, although as the number of samples increased its success rate showed improvement.

7) Characterization of resulting policies: While conducting the above experiments we noticed that in the majority of cases, the resulting controller would either succeed in balancing the pendulum all the time or would fail immediately all the times. Controllers exhibiting a mixed behavior were very rare. This is a very desirable property to have; it means that with little testing, one can get a good idea about the quality of a particular learned controller.

B. Bicycle Balancing and Riding

The goal in the Bicycle Balancing and Riding problem [17] is to learn to balance and ride a simulated bicycle to a target position located 1 km away from the starting location. Initially, the bicycle's orientation is at an angle of 90° to the goal. The state space is a 5-dimensional continuous vector (θ, $\dot{\theta}$, ω, $\dot{\omega}$, ψ), where θ is the angle of the handlebar, ω is the vertical angle of the bicycle, and ψ is the angle of the bicycle to the goal. The actions are the torque τ applied to the handlebar and the displacement υ of the rider. The original discrete action space allowed the values $(-2, 0, +2)$ for τ and the values $(-0.02, 0, 0.02)$ for υ, yielding a total of 9 discrete joint actions. Our continuous action space allows any value within these ranges, therefore $\tau \in [-2, +2]$ and $\upsilon \in [-0.02, 0.02]$. The state vector was augmented to include the last displacement of the rider and the last torque applied to the handlebar and becomes a 7-dimensional vector (θ, $\dot{\theta}$, ω, $\dot{\omega}$, ψ, τ, υ). The modification policy makes a decision to increase or decrease each action variable, giving a total of 4 discrete joint actions ({-1,-1}, {-1,1}, {1,-1}, {1,1}). The noise in the system is a uniformly distributed term in $[-0.02, +0.02]$ added to the displacement component of the action. The time step of the simulation was set to 0.01 seconds and $\gamma = 0.8$. If the vertical angle ω exceeds $\pi/15$ in absolute value, the bicycle falls and the episode terminates.

The state-action value function Q was approximated by a linear combination of 29 polynomial basis functions

$$\phi = \Big(1, \omega, \dot{\omega}, \omega^2, \dot{\omega}^2, \omega\dot{\omega}, \upsilon, \upsilon\omega, \upsilon\dot{\omega}, \upsilon\omega\dot{\omega}, \theta, \dot{\theta}, \theta^2, \dot{\theta}^2, \theta\dot{\theta}, \tau,$$
$$\tau\theta, \tau\dot{\theta}, \tau\theta\dot{\theta}, \omega\theta, \omega\theta^2, \omega^2\theta, \tau\upsilon, \psi, \psi^2, \psi\theta, \bar{\psi}, \bar{\psi}^2, \bar{\psi}\theta\Big)^\top,$$

where $\bar{\psi} = \pi - \psi$ for $\psi > 0$ and $\bar{\psi} = -\pi - \psi$ for $\psi < 0$. This block of 29 basis functions was repeated for each of the 4 discrete joint actions, giving a total of 116 basis functions. The following shaping reward was used:

$$r = -\left(\beta_1 \frac{|\omega|}{(\pi/15)} + \beta_2 \frac{\Delta distance}{speed \times \Delta t}\right)$$

where $\Delta distance$ is the change in the distance of the bicycle to the goal during the last step, $speed$ is the constant linear speed of the bicycle, and Δt is the time step. The first component (weighted by β_1) is normalized to $[0, 1]$ and penalizes large vertical angles for the bicycle to encourage successful balancing. The second component (weighted by β_2) is normalized to $[-1, 1]$ and rewards progress towards the goal and penalizes movement away from it to encourage successful riding to the goal. The weights of the two components were determined empirically to balance these two occasionally conflicting objectives; the values used were $\beta_1 = 16$ and $\beta_2 = 1.2$. Note that the final reward is a single value and the contribution of each component cannot be distinguished. Samples for learning were collected using a purely random policy; each episode was allowed to run for a maximum of 15 steps. Under a random policy, the remaining steps of each episode until termination (about 45 more steps) simply cover an uncontrollable course to a crash, where actions have no impact. Therefore, these steps were ignored as useless and potentially problematic, since they would bias the sample distribution.

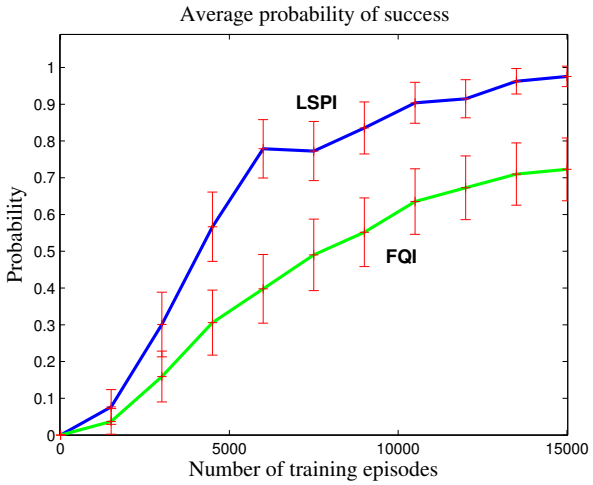

Fig. 7. Bicycle : Average probability of success.

Figure 7 shows the performance of the learned controllers as a function of the number of training episodes. For each batch of training episodes, the learned policy was evaluated 100 times in order to estimate accurately the probability of success and the average number of balancing steps. This experiment was repeated 100 times for the entire horizontal axis, to obtain average results over different sample sets and the 95% confidence intervals. Each episode was allowed to run for a maximum of 72,000 steps corresponding to 2 kilometers of riding distance. Episodes that reached the goal within this limit were considered to be successful. As the number of training episodes increases, the resulting policies become consistently more successful. For 15,000 training episodes (225,000 samples) the expected probability of success for LSPI is 97.56%, the expected probability of crashing is 1.38%, and the expected number of balancing steps is 71,231. Fitted Q-Iteration did not perform as well in this domain reaching an expected probability of success of 72.4% for 15,000 training episodes. Still, it clearly shows a rate of improvement with more samples. It is also likely that with a different (possibly, non-linear) approximation architecture its success rate would be comparable, or even better, than that of LSPI. It is interesting to note, that once again the majority of policies either succeed all the time or fail all the time.

VI. Discussion

A. Strengths and Weaknesses

The proposed method offers several advantages:

- It is simple and easy to implement, both conceptually and in terms of the amount of code required.
- It requires little or no tuning. Once the desired resolution is decided, the only parameter left to tune is K, which can be simply set to 2.

- It requires only a binary decision on behalf of the underlying reinforcement learning algorithm for each action variable. This allows for very fast and efficient implementations of the entire policy function.
- It easily achieves resolutions impossible to reach with discrete actions and better than what is practically possible with popular approximation techniques.
- It can be used in conjunction with any reinforcement learning algorithm that supports discrete actions and continuous states and can be used in an online, offline, on-policy, or off-policy setting.
- It has low computational requirements. It requires only a small number of additions, subtractions, shift operations, and conditional statements. This fact makes it perfect for embedded platforms with limited resources.
- It offers better scalability properties than most (if not, all) existing continuous-action RL algorithms, when the number of action variables increases.
- It produces policies which exhibit a "minimum required effort" behavior which is vital for critical resources.
- It results in smooth control, while being able to respond quickly when the situation requires it.

Of course, everything comes at a cost. As mentioned before, this technique relies on the temporal locality property of actions. In quickly changing domains, we may have to decrease the time step (increase the rate at which our controller makes decisions). Obviously in domains that exhibit no temporal locality, this method is not suitable. Also the state space of the problem that we are trying to solve is now more complex. It includes the original state variables and three new state variables: A, Δ and e. As shown, adding only A to the state vector is sufficient for most problems. Although it definitely puts a strain to the underlying learning algorithm, its overhead is far less than naïvely increasing the number of possible decisions beyond a certain point. In summary, we strongly believe that the benefits of the proposed approach far outweigh its costs and make it a viable choice for learning continuous-action control policies.

B. Future Work

In all applications presented in this paper, the action range is partitioned into equally-sized intervals. On a number of applications there may be areas of the action space that require finer resolution, while others are less important. Therefore, a skewing function could be used, to distort the action space, creating a more appropriate fit for the available resolution. There may be domains where using more than two actions in the discrete policy would be more appropriate than decreasing the time step. In such cases one could have four or more actions, where two will increase/decrease the continuous variable by a factor Δ_1, two by a factor Δ_2, and so on. The Δ update equation used here is not the only valid rule for adaptive schemes. Other schemes, possibly with different growth and decay rates, offering better stability or other desirable properties, are not hard to develop. A large amount of research in other fields has been devoted to finding ways of representing analog (continuous) signals with digital (discrete) means. Many of these results could potentially be adopted, providing insight from mature areas to developing ones.

VII. Conclusion

Inspired by methods used in telecommunications systems, this paper introduced a generic approach for learning continuous-action control policies in domains exhibiting temporal action locality. The computational efficiency and scalability properties of the algorithm have been carefully addressed and its applicability has been demonstrated experimentally in two domains in conjunction with three known RL algorithms. It is our belief that the proposed scheme will enable RL researchers to broaden their application domains and extend their favorite RL algorithms to a variety of practical real-world control problems.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[2] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, Cambridge, United Kingdom, 1989.

[3] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[4] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, no. 2-3, pp. 161–178, 2002.

[5] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[6] J. G. Proakis and M. Salehi, *Communication Systems Engineering*. Prentice Hall, 2001.

[7] C. Gaskett, D. Wettergreen, and E. Zelinsky, "Q-learning in continuous state and action spaces," in *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*. Springer-Verlag, 1999, pp. 417–428.

[8] H. M. Gross, V. Stephan, and M. Krabbes, "A neural field approach to topological reinforcement learning in continuous action spaces," in *Proceedings of the International Joint Conference on Neural Networks*, 1998, pp. 1992–1997.

[9] T. Strösslin and W. Gerstner, "Reinforcement learning in continuous state and action space," in *International Conference on Artificial Neural Networks*, 2003.

[10] C. Touzet, "Neural reinforcement learning for behaviour synthesis," *Robotics and Autonomous Systems*, vol. 22, pp. 251–281, 1997.

[11] A. Lazaric, M. Restelli, and A. Bonarini, "Reinforcement learning in continuous action spaces through sequential monte carlo methods," in *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press, 2008, pp. 833–840.

[12] A. O. Esogbue and W. E. Hearnes, "A learning algorithm for the control of continuous action set-point regulator systems," *Journal of Computational Analysis and Applications*, vol. 1, no. 2, pp. 121–234, 1999.

[13] B. Sallans and G. E. Hinton, "Reinforcement learning with factored states and actions," *Journal of Machine Learning Research*, vol. 5, pp. 1063–1088, 2004.

[14] J. C. Santamaría, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive Behavior*, vol. 6, pp. 163–218, 1998.

[15] M. Riedmiller, "Application of a self-learning controller with continuous control signals based on the DOE-approach," in *Proceedings of the European Symposium on Neural Networks*, 1997.

[16] H. O. Wang, K. Tanaka, and M. F. Griffin, "An approach to fuzzy control of nonlinear systems: Stability and design issues," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 1, pp. 14–23, 1996.

[17] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of The Fifteenth International Conference on Machine Learning*, 1998, pp. 463–471.