Learning to Control a 6-Degree-of-Freedom Walking Robot

Paweł Wawrzyński
Warsaw University of Technology, Institute of Control and Computation Engineering, Warsaw, Poland
Email: p.wawrzynski@elka.pw.edu.pl

***Abstract*— We analyze the issue of optimizing a control policy for a complex system in a simulated trial-and-error learning process. The approach to this problem we consider is Reinforcement Learning (RL). Stationary policies, applied by most RL methods, may be improper in control applications, since for time discretization fine enough they do not exhibit exploration capabilities and define policy gradient estimators of very large variance. As a remedy to those difficulties, we proposed earlier the use of piecewise non-Markov policies. In the experimental study presented here we apply our approach to a 6-degree-of-freedom walking robot and obtain an efficient policy for this object.**

***Keywords*— Machine Learning, Reinforcement Learning, Continuous Time, Adaptive Control.**

I. Introduction

Let us consider the issue of developing a control policy for a complex system. Usually the more complex is the system, the more difficult is the task of designing its optimal controller. However, simulating dynamics of this system may still be tractable. The idea is then to build an appropriate simulator and develop a close to optimal controller in a simulated trial-and-error learning process.

A particular approach to trial-and-error control optimization is Reinforcement Learning (RL). In general, this area considers the issue of an intelligent agent that optimizes its behavior in an initially unknown environment. We would like to construct controllers that are able to "learn" by trial and error to control plants whose dynamics are unknown. The controller may be understood as the agent, which is rewarded for reaching the control objectives.

Most RL algorithms [1], [2], [3], [4], [5] optimize stationary policies, i.e. ones that draw an action only on the basis of a current state. The stationary policies may be improper in learning control issues because of two reasons. First, under some quite typical conditions randomness in the overall agent-environment system diminishes as the time discretization is becoming finer. Hence, the agent's exploratory behavior does not give any clues to policy improvement [6]. Second, keeping variance of policy gradient estimators bounded when the time discretization is becoming finer may require increasing randomness in the optimized policy which may make the policy unfeasible [7]. In order to overcome those difficulties we propose the use of piecewise non-Markov policies. It is shown in [7] that a given Markov Decision Process (MDP) in tandem with a piecewise non-Markov policy define a new MDP and a stationary policy. By applying an ordinary RL algorithm to this new MDP we can optimize the initial non-Markov policy. It is also demonstrated that combination of the new MDP with some existing RL methods may be understood as their enhancement. These methods include all algorithms based on likelihood ratio, i.e. Episodic REINFORCE [8] and derivative algorithms such as OLPOMDP [9], the method introduced in [10], and others.

In the experimental study we apply our approach to optimize a control policy of a simulated robot. The action space in our problem is 6-dimensional and the state space is 31-dimensional. While problems of such complexity has been solved with the use of RL algorithms [5], [11], these algorithms were applied there only to optimize a certain level of a hierarchical policy. In our study RL is applied to optimize a flat policy of a complex system.

The paper is organized as follows. In Sec. II the problem of our interest is presented along with difficulties in learning a stationary policies in fine time discretization. The next section presents the remedy to those problems in the form of non-Markov policies. In Sec. IV an example of such a policy is discussed. Section V our approach is verified empirically with the use of 6-degree-of-freedom walking robot. Sec. VI contains a discussion of the the obtained results and another section concludes. This paper continues the work [7] to which the reader is often referred.

II. Problem Formulation

We will consider the standard episodic RL setup [12]. A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, P_s, r, P_0, \mathcal{S}^* \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively; $\{P_s(\cdot|s,a) : s \in \mathcal{S}, a \in \mathcal{A}\}$ is a set of state transition probabilities; we write $s_{t+1} \sim P_s(\cdot|s_t, a_t)$. In this work we assume that both $\mathcal{S}$ and $\mathcal{A}$ are multidimensional continuous and each P_s is a density. The immediate

reward, r_t, depends on the action and the next state, $r_t = r(a_t, s_{t+1})$. P_0 is the distribution of first states of each episode and $\mathcal{S}^*$ is the set of terminal states. The objective of a learning process is to find a control policy that maximizes future rewards in each state.

We are interested in applications of the solution of the above RL problem to learning control tasks. A painful difficulty that emerges in control problems is the fine time discretization. It makes a single action impact the overall performance insignificantly. Furthermore, the impact of the action emerges a large number (thousands) of steps after the very action took place. We require the learning algorithm to work properly no matter how fine the time discretization is.

RL methods are usually designed to optimize a stationary policy, i.e. one that draws in each instant an action, a_t, on the basis of the present state, s_t, and the policy parameter, θ, namely

$$a_t \sim \pi(\cdot\,; s_t, \theta).$$

In this section we analyze by means of a simple example, how the time discretization influences policy gradient estimation when a policy is defined that way. An important insight to this issue has been provided in [6] where it has been shown that in a continuous environment, under quite general conditions, the state trajectory converges to a deterministic limit as the time discretization diminishes. The question arise how this phenomenon influences quality of policy gradient estimators. A general answer to this question is difficult to provide. However, the simple example below suggests that this influence can be damaging.

Let state represent one-dimensional velocity and action represent one-dimensional acceleration. We have $\mathcal{S} = \mathcal{A} = \Re$. An episode lasts for 1 sec. and it includes T steps, $\delta = 1/T$ long each. Within each step an action is drawn from the normal distribution $N(\theta, \sigma_a^2)$ where θ is a policy parameter. The action defines constant acceleration within a step and velocity at the beginning of a trial is null. The only nonzero reward is equal to noised velocity in the last state. We have

$$s_t = \begin{cases} 0 & \text{iff } t = 0 \\ s_{t-1} + \delta a_t & \text{iff } t > 0, \end{cases}$$
$$r_t = \begin{cases} 0 & \text{iff } t < T-1 \\ s_T + y_T & \text{iff } t = T-1, \end{cases}$$

where y is a random variable drawn from the normal distribution $N(0, \sigma_y^2)$.

The quality index, $J(\theta)$, of the policy defined by θ is equal to the expected reward at the end of an episode. Because the final reward is a sum of random variables, we have

$$\mathcal{E}_\theta r_{T-1} = \mathcal{E}_\theta \left(\sum_{t=0}^{T-1} \delta a_t + y_T \right) = T\delta\theta = \theta.$$

Therefore, $J(\theta) = \theta$ and $\nabla J(\theta) = 1$. Our main concern here is variance of the policy gradient estimator. We will consider the policy gradient estimator applied in the episodic REINFORCE algorithm [8] since prevailing policy gradient estimators can be considered modifications of this early formula. The estimator applied to our problem is of the form

$$\widehat{g} = (r_{T-1} - c) \sum_{t=0}^{T-1} \frac{\partial \ln \pi(a_t; s_t, \theta)}{\partial \theta^T}$$

where c is the *baseline*. Variance of this estimator is defined by the following formula

$$\mathcal{V}\widehat{g} = 2 + \frac{1}{\delta\sigma_a^2}\left((c-\theta)^2 + \sigma_y^2\right) \tag{1}$$

derived in [7]. We can see that the larger action variance, the smaller gradient estimator variance. The exploration–exploitation balance becomes conspicuous when we compare $\mathcal{V}\widehat{g}$ with variance of s_T, namely $\mathcal{V}s_T = \delta\sigma_a^2$. By comparing this value with (1), we can see that $\mathcal{V}s_T$ is "almost" inversely proportional to $\mathcal{V}\widehat{g}$.

What happens when the time discretization parameter δ decreases? In order to keep gradient estimator variance small, variance of action has to be increased. In fact, variance of gradient estimator remains constant if only

$$\sigma_a^2 \propto 1/\delta.$$

Notice that this way variance of s_T is also stabilized.

It is seen in our example that in order to keep variance of policy gradient estimator bounded while the time discretization decreases, we have to increase variance of action. However, "actions" in control systems are always bounded; hence, they can not have too large variance either. Therefore, while fine time discretization is necessary for good control it contradicts with quality of policy gradient estimation.

III. MDP Defined by Non-Markov Periods

Let the policy applied by the agent be *piecewise non-Markov* in the following sense. It divides an episode into periods and generates actions within each period on the basis of previous actions and states in this period. Let the periods be indexed by k and k-th period starts at time t_k and lasts for l_k instants. For $i : 0 \le i < l_k$ we have[1]

$$a_{t_k+i} \sim \pi(\cdot\,; s_{t_k}, a_{t_k}, \ldots, s_{t_k+i}, \theta). \tag{2}$$

[1]We apply "..." also to denote a subsequence of the sequence $(s_1, a_1, s_2, a_2, \ldots)$.

Let the periods defined by a piecewise non-Markov policy be called *non-Markov periods* or, in short, *nm-periods*.

Given a Markov Decision Process $M = \langle \mathcal{S}, \mathcal{A}, P_s, r, P_0, \mathcal{S}^* \rangle$ we define a new one, $\bar{M} = \langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{P}_s, \bar{r}, P_0, \mathcal{S}^* \rangle$ with the use of nm-periods defined above. Let states and actions in $\bar{M}$ be denoted by $\bar{s}$ and $\bar{a}$, respectively, and time be indexed by k. Simultaneously, given a piecewise non-Markov policy π in M, we define a stationary policy $\bar{\pi}$ in $\bar{M}$ generating actions from $\bar{\mathcal{A}}$. States in $\bar{M}$ correspond to first states in nm-periods; we have

$$\bar{s}_k = s_{t_k} \quad \text{and} \quad \bar{\mathcal{S}} = \mathcal{S}.$$

Actions in $\bar{M}$ correspond to joint trajectories of states and actions within nm-periods, namely

$$\bar{a}_k = \langle a_{t_k}, s_{t_k+1}, \ldots, a_{t_k+l_k-1} \rangle \text{ and } \bar{\mathcal{A}} = \bigcup_{i \geq 0} \mathcal{A} \times (\mathcal{S} \times \mathcal{A})^i.$$

The transition distribution in $\bar{M}$, denoted by $\bar{P}_s$, is defined by P_s, π, and the way l_k emerges. In the simplest case $l_k = l$ for a certain constant l unless k-th period is the last one in the episode; then $1 \leq l_k \leq l$. We are free to define the method of calculating rewards in $\bar{M}$. For instance, a reward in $\bar{M}$ can be the average value of rewards gathered within the corresponding nm-period in M. The distribution of first states P_0 and the set of terminal states $\mathcal{S}^*$ remain unchanged.

An action in $\bar{M}$ is generated by the policy π in tandem with P_s. What is yet important is that we can calculate the likelihood ratio $\nabla_\theta \ln \bar{\pi}(\bar{a}_k; \bar{s}_k, \theta)$. Let us denote

$$\begin{aligned} S_k &= [s_{t_k}^T, \ldots, s_{t_k+l_k-1}^T]^T, \\ A_k &= [a_{t_k}^T, \ldots, a_{t_k+l_k-1}^T]^T, \\ \pi_A(A_k; S_k, \theta) &= \prod_{i=0}^{l_k-1} \pi(a_{t_k+i}; s_{t_k}, a_{t_k}, \ldots, s_{t_k+i}, \theta). \end{aligned} \tag{3}$$

π_A is a density of a sequence of actions within an nm-period given a sequence of states. It is entirely defined by the way the policy π generates actions. From the decomposition

$$\begin{aligned} \bar{\pi}(\bar{a}_k; \bar{s}_k, \theta) = \prod_{i=0}^{l_k-1} \pi(a_{t_k+i}; s_{t_k}, a_{t_k}, \ldots, s_{t_k+i}\theta) \times \\ \times \prod_{i=0}^{l_k-2} P_s(s_{t+i+1} | s_{t_k+i}, a_{t_k+i}) \end{aligned}$$

we see that

$$\nabla_\theta \ln \bar{\pi}(\bar{a}_k; \bar{s}_k, \theta) = \nabla_\theta \ln \pi_A(A_k; S_k, \theta).$$

A special feature of $\bar{M}$ is the fact that the agent is not entirely free to choose an action from $\bar{\mathcal{A}}$. It is hence impossible to apply *Q-Learning* [2] or SARSA to $\bar{M}$. However, optimization of a stationary policy in $\bar{M}$ can be in principle performed by all methods based on likelihood ratio, including episodic REINFORCE [8], Actor-Critics [3], [4], [5], OLPOMDP [9] and others.

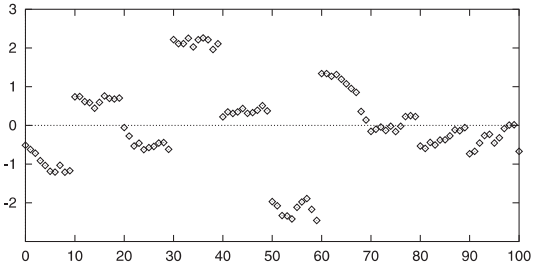

Fig. 1. A run of the piecewise independent autoregressive process, ξ_t, for $\Sigma = 1, \alpha = 0.99, t_{k+1} - t_k \equiv 10$. Within an nm-period there is a correlation between random elements while there is no correlation between elements in different nm-periods.

IV. Piecewise Non-Markov Policies, An Example

In the present section we define a simple class of non-Markov policies. These policies exploit each k-th period to carry out a coherent experiment that provides a clue to an improvement of the policy. This coherence is a consequence of the fact that while at each moment the action has a random component, there is a stochastic dependence among these components within the same nm-period. In the next subsection we analyze a way of generating such stochastically dependent components.

Piecewise independent autoregressive process: Let $\{\epsilon_t, t = 1, 2, \ldots\}$ be a sequence of independent random vectors in $\mathfrak{R}^n$ drawn from the normal distribution with zero mean and covariance matrix Σ, i.e. $N(0, \Sigma)$. Also, let $\alpha \in (0, 1)$ and $\{\xi_t, t = 1, 2, \ldots\}$ be a sequence of random vectors in $\mathfrak{R}^n$ computed as

$$\xi_t = \begin{cases} \epsilon_t & \text{if } t = t_k \text{ for any } k \\ \alpha \xi_{t-1} + \sqrt{1 - \alpha^2} \epsilon_t & \text{otherwise.} \end{cases} \tag{4}$$

The process defined above has the following features derived in [7]:

- All ξ_t are of the normal distribution $N(0, \Sigma)$.
- For t and t' in different nm-periods,

$$\operatorname{cov}(\xi_t, \xi_{t'}) = \mathcal{E}(\xi_t \xi_{t'}^T) = 0$$

- for t and t' in the same nm-period,

$$\operatorname{cov}(\xi_t, \xi_{t'}) = \alpha^{|t'-t|} \Sigma. \tag{5}$$

Consequently, $\{\xi_t, t = t_k, \ldots, t_{k+1} - 1\}$ happens to be an autoregressive stochastic process. Notice that $\operatorname{cov}(\xi_t, \xi_t) \equiv \Sigma \equiv \operatorname{cov}(\epsilon_t, \epsilon_t)$.

The random process we defined above, the piecewise independent autoregressive process, may be interpreted as a simple method of transforming normal white noise, ϵ_t, into sequences of random vectors that are stochastically

dependent (see Fig. 1). Thank to that dependence, each of these sequences may support a coherent experiment that gives a clue to policy improvement. Below we present a policy that makes use of such experiments.

Deterministic Transformation + Stochastic Process: Let $\mathcal{A} = \Re^{n_a}$ and an action, a_t, be calculated as

$$a_t = \widetilde{a}(s_t; \theta) + \xi_t \tag{6}$$

where $\widetilde{a} : \mathcal{S} \times \Theta \mapsto \mathcal{A}$ is a certain function, e.g. a neural network with input s and weights θ. We will denote by $\nabla\widetilde{a}(s, \theta)$ a matrix of derivatives of $\widetilde{a}$ with respect to its second argument, namely

$$\nabla\widetilde{a}(s, \theta) = \frac{\partial\widetilde{a}(s, \theta)}{\partial\theta^T} = \left[\frac{\partial\widetilde{a}_j(s, \theta)}{\partial\theta_i}\right]_{i,j}.$$

What we need is to define the distribution $\pi_A(A_k; S_k, \theta)$ and the likelihood ratio $\nabla_\theta \ln \pi_A(A_k; S_k, \theta)$. for S_k and A_k defined in (3). Given trajectory S_k, quantities a_t result from adding random elements ξ_t to constant values $\widetilde{a}(s_t; \theta)$. Consequently, the distribution $\pi_A(A_k; S_k, \theta)$ is the normal one with the expected value and the covariance matrix equal to

$$\mathcal{E}A_k = m_k(\theta) = \begin{bmatrix} \widetilde{a}(s_{t_k}, \theta) \\ \vdots \\ \widetilde{a}(s_{t_k+l_k-1}, \theta) \end{bmatrix},$$

$$\operatorname{cov} A_k = C_k = \begin{bmatrix} \Sigma & \alpha\Sigma & \cdots & \alpha^{l_k-1}\Sigma \\ \alpha\Sigma & \Sigma & & \\ \vdots & & \ddots & \vdots \\ \alpha^{l_k-1}\Sigma & & \cdots & \Sigma \end{bmatrix},$$

respectively. $\operatorname{cov} A_k$ results from the fact that $\operatorname{cov}(a_t, a_{t'}) = \operatorname{cov}(\xi_t, \xi_{t'})$ and (5).

We thus deal with the normal distribution $N(\mathcal{E}A_k, \operatorname{cov} A_k)$ whose mean depends on the parameter θ and variance does not. The density of this distribution is given by

$$\pi_A(A; S_k, \theta) = \left(\sqrt{2\pi}^{l_k n_a} |C_k|\right)^{-1} \times \tag{7}$$
$$\times \exp\left(-0.5(A - m_k(\theta))^T (C_k)^{-1}(A - m_k(\theta))\right).$$

We also have

$$\nabla_\theta \ln \pi_A(A; S_k, \theta) = (\nabla_\theta m_k(\theta))\, C_k^{-1}(A - m_k(\theta))$$
$$= [\nabla\widetilde{a}(s_{t_k}; \theta) \cdots \nabla\widetilde{a}(s_{t_k+l_k-1}; \theta)] C_k^{-1}(A - m_k(\theta)).$$

In order to apply the above equation directly, one would have to invert the matrix C_k which is an operation of complexity $O(n^4)$ for $n = \dim A_k = l_k n_a$. A cheaper method is to compute vector $(C_k)^{-1}(A - m_k(\theta))$ as y satisfying the linear equation

$$C_k y = A - m_k(\theta).$$

This operation is of complexity $O(n^3)$ which still can be inconvenient for large n. Fortunately, the matrix C_k^{-1} has a known compact form. In order to express it, let us denote for matrices X and Y, a term $X \otimes Y$ as the matrix

$$X \otimes Y = \begin{bmatrix} X_{1,1}Y & X_{1,2}Y & \cdots \\ X_{2,1}Y & X_{2,2} & \\ \vdots & & \ddots \end{bmatrix}.$$

One easily shows that

$$(X \otimes Y)^{-1} = X^{-1} \otimes Y^{-1}.$$

Moving back to the problem of computing C_k^{-1}, let us denote by Λ_k an $(l_k \times l_k)$-matrix of the form

$$\Lambda_k = [\alpha^{|i-j|}]_{i,j}.$$

Obviously

$$C_k = \Lambda_k \otimes \Sigma.$$

The matrix Λ_k^{-1} has a compact form. For $l = 1$ it is an identity matrix. However, for $l > 1$,

$$\Lambda_k^{-1} = \frac{1}{1-\alpha^2} \begin{bmatrix} 1 & -\alpha & & & 0 \\ -\alpha & 1+\alpha^2 & & & \\ & -\alpha & \ddots & -\alpha & \\ & & & 1+\alpha^2 & -\alpha \\ 0 & & & -\alpha & 1 \end{bmatrix}.$$

Therefore,

$$C_k^{-1}(A - m_k(\theta)) = (\Lambda_k^{-1} \otimes \Sigma^{-1})(A - m_k(\theta)).$$

If, Σ is diagonal, which is an obvious choice, computation of the above vector can be easily implemented as an operation of complexity $O(n)$.

Problems with deficit of computational power. Sampling: The size of the matrix $\operatorname{cov} A$ and the vector $\nabla_\theta \ln \pi_A(A; S, \theta)$ grows with the number of actions in the non-Markov period. Therefore, in control applications with very fine time discretization, the matrix and the vector may be very large and their computation may be very power consuming. A certain solution to this problem may be the sampling—a technique applied successfully in a number of areas, e.g. in sound processing. In order to have a good idea of the content of a piece of sound, it is enough to have samples that evenly covers the piece.

Let us denote by $\widehat{A}$ a vector comprised of every Δ-th action

$$\widehat{A} = [a_1^T\ a_{1+\Delta}^T\ a_{1+2\Delta}^T \cdots a_L^T]^T$$

where $L = 1 + \lfloor (l-1)/\Delta \rfloor \Delta$ is the last sampling instant. If a trajectory of actions, A, is regular, then $\widehat{A}$ is its approximate representation and by increasing/decreasing the probability of $\widehat{A}$ we also increase/decrease the probability of A. Therefore

$$\nabla_\theta \ln \pi_{\widehat{A}}(\widehat{A}; S, \theta)$$

may be regarded as an approximation of

$$\nabla_\theta \ln \pi_A(A; S, \theta).$$

Also, we have

$$\nabla_\theta \ln \pi_{\widehat{A}}(\widehat{A}; S, \theta) = [\nabla \widetilde{a}(s_1, \theta) \ \cdots \ \nabla \widetilde{a}(s_L, \theta)] \times \\ \times (\operatorname{cov} \widehat{A})^{-1} (\widehat{A} - \mathcal{E}\widehat{A}).$$

Notice that the dimensionality of $\widehat{A}$ is for $\Delta > 1$ smaller than that of A. Therefore, operations on $\nabla_\theta \ln \pi_{\widehat{A}}(\widehat{A}; S, \theta)$ may be much less time consuming than those on $\nabla_\theta \ln \pi_A(A; S, \theta)$.

When choosing appropriate Δ, one has to balance available computational power (the larger Δ the less exploited it is) and the quality of approximation offered by $\nabla_\theta \ln \pi_{\widehat{A}}(\widehat{A}; S, \theta)$ (the larger Δ, the worse it is).

V. Experiments with Half-Cheetah

We apply the presented approach to optimize a policy for a planar model of a walking animal. At this point we refer to a large body of research on walking robots. Examples of these are studies on locomotion of dog-like robot [13], [14], [15], hexapod [16], salamander-like robot [17], or human-like biped [18], [11]. Control systems for such robots are usually developed on the basis of a large amount of preliminary knowledge that leave little room for adaptation or learning. However, in [11], Natural Actor-Critic is applied to find parameters for Central Pattern Generator that controls a bipedal simulated robot. In our experiment an RL algorithm is employed to directly optimize a non-Markov policy.

A. Object Description

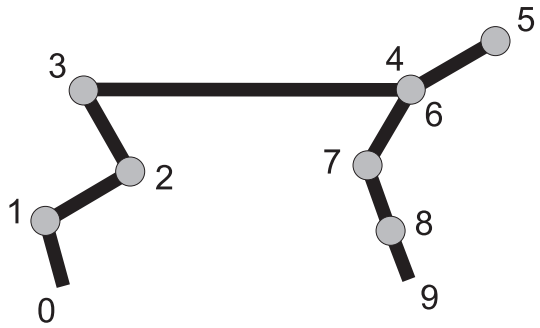

Fig. 2. Half-Cheetah.

Half-Cheetah, presented in Fig. 2, is a planar kinematic string of 9 links and 10 joints; the "paws" of Half-Cheetah will also be called joints. The angles of 4-th and 5-th joint are fixed, all the the others are controllable. Consequently, Half-Cheetah is a 6-degree-of-freedom walking robot. The fact that it is designed as a kinematic string allows efficient simulation of its dynamics, e.g. using the Newton-Euler method [19]. Mass of the object is concentrated in its joints which weigh

$$1,\ 1,\ 1,\ 4,\ 1,\ 2,\ 1,\ 1,\ 1,\ \text{and } 1$$

kilos. Consecutive links of Half-Cheetah are of lengths

$$\frac{0.2}{\cos\frac{\pi}{12}},\ \frac{0.15}{\cos\frac{\pi}{3}},\ \frac{0.25}{\cos\frac{\pi}{6}},\ 1,\ 0.3,\ 0.3,\ \frac{0.25}{\cos\frac{\pi}{9}},\ \frac{0.2}{\cos\frac{\pi}{9}},\ \frac{0.15}{\cos\frac{\pi}{9}}$$

meters. The initial angle between the first link and the perpendicular is $-\pi/12$. The initial angles of the joints, ω_i^0 for $i = 1, \ldots, 8$, are

$$-\frac{5}{12}\pi,\ \frac{1}{2}\pi,\ -\frac{2}{3}\pi,\ \frac{1}{6}\pi,\ -\pi,\ \frac{1}{6}\pi,\ \frac{5}{18}\pi,\ \text{and } 0.$$

The movement of joints are limited to intervals. Their lower bounds are

$$-\frac{5}{6}\pi,\ \frac{1}{4}\pi,\ -\frac{5}{6}\pi,\ \frac{1}{6}\pi,\ -\pi,\ \frac{1}{18}\pi,\ 0,\ -\frac{2}{3}\pi,$$

and the upper ones are

$$-\frac{1}{6}\pi,\ \frac{3}{4}\pi,\ -\frac{1}{3}\pi,\ \frac{1}{6}\pi,\ -\pi,\ \frac{1}{2}\pi,\ \frac{8}{9}\pi,\ \frac{1}{9}\pi.$$

The object is controlled by applying torques at the joints. The torch τ_i at i-th joint is computed in two phases: the variable

$$\tau_i^0 = 2\pi^{-1}\text{arctg}\left(-2(\omega_i - \omega_i^0) - 0.05\dot{\omega}_i\right)$$

expresses "spontaneous" torch at i-th joint implied by its angle, ω_i, and angular velocity, $\dot{\omega}_i$. It is added to the signal a_i^0 coming from the learning controller, projected on the interval $[-1, 1]$, and multiplied by the "strength", T_i, of i-th joint, namely

$$\tau_i = T_i \min\left\{\max\{-1, \tau_i^0 + a_i^0\}, 1\right\}.$$

The "strength" are of values

$$60,\ 90,\ 120,\ -,\ -,\ 90,\ 60,\ 30.$$

The objective of control is to make Half-Cheetah run forward as fast as possible. Notice that without the external control (i.e. for $a_i^0 \equiv 0$), the torch $T_i\tau_i^0$ implements a PD control[2] with saturation. It stabilizes the joint angle close to its initial value which makes Half-Cheetah stand still.

B. State and reward

In order to apply reinforcement learning to Half-Cheetah, one has to define state and reward the learning algorithm has access to. In the below specification of these entities the joints of the object have indexes from 0 to 9 whereas the links have indexes from 0 to 8. The 2-dimensional position of i-th joint is denoted by (x_i, y_i) and its speed by $(\dot{x}_i, \dot{y}_i)$. The angle between i-th link and the horizontal line is denoted by ω_i^l and its angular speed

[2] A differentiator with proportional gain.

TABLE I

VARIABLES OF HALF-CHEETAH STATE. $j = 0, 1, 2, 3$ (BEFORE THE NECK) AND $k = 0, 1, 2$ (AFTER THE NECK)

$$
\begin{aligned}
s_{t,1} &= (y_0 - 0.1)/0.1 \\
s_{t,2} &= \dot{y}_0/0.3 \\
s_{t,3} &= ([y_0 = 0] - 0.5)/0.5 \\
s_{t,4} &= ((y_3 + y_4)/2 - 0.6)/0.2 \\
s_{t,5} &= (\dot{x}_3 + \dot{x}_4)/2 - 1 \\
s_{t,6} &= \dot{y}_3/0.3 \\
s_{t,7} &= \dot{y}_4/0.3 \\
s_{t,8} &= (y_9 - 0.1)/0.1 \\
s_{t,9} &= \dot{y}_9/0.3 \\
s_{t,10} &= ([y_9 = 0] - 0.5)/0.5 \\
s_{t,11+3j} &= \cos\omega_j^l/0.7 \\
s_{t,12+3j} &= \sin\omega_j^l/0.7 \\
s_{t,13+3j} &= \dot{\omega}_j^l/3 \\
s_{t,23+3k} &= \cos\omega_{6+k}^l/0.7 \\
s_{t,24+3k} &= \sin\omega_{6+k}^l/0.7 \\
s_{t,25+3k} &= \dot{\omega}_{6+k}^l/3
\end{aligned}
$$

TABLE II

THE CLASSIC ACTOR-CRITIC ALGORITHM WITH TD(λ)-ERRORS AND NON-MARKOV PERIODS.

1. At k-th nm-period, perform l_k actions according to (2), (4), and (6). Register the states S_k, the actions A_k and the aggregated reward $\bar{r}_k$.
2. If the trial has been artificially stopped at k-th period, set $k := k + 1$ and repeat from Step 1.
3. Compute the temporal difference
$$d_k = \bar{r}_k + \gamma_k \widetilde{V}(s_{t_{k+1}}; \upsilon) - \widetilde{V}(s_{t_k}; \upsilon).$$
4. Policy improvement
$$z := \lambda\gamma_k z + \nabla_\theta \ln \pi_{\widehat{A}}(\widehat{A}_k; S_k, \theta),$$
$$\theta := \theta + \beta_k^\theta d_k z.$$
5. Critic improvement
$$y := \lambda\gamma_k y + \frac{\partial}{\partial \upsilon^T}\widetilde{V}(s_{t_k}; \upsilon),$$
$$\upsilon := \upsilon + \beta_k^\upsilon d_k y.$$
6. Set $k := k + 1$ and repeat from Step 1.

by $\dot{\omega}_i^l$. Each state variable is normalized to roughly cover the interval $[-1, 1]$. We also apply the notation

$$[predicate] = \begin{cases} 1 & \text{if } predicate \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

E.g. the term $[y_0 = 0]$ is equal to 1 if the hind paw stands on the ground and 0 otherwise. All 31 variables describing state of Half-Cheetah are defined in Table I.

In our definition of reward we apply a *soft-step function* of the form

$$f(x) = \begin{cases} 0 & \text{for } x \le 0 \\ 2x^2 & \text{for } x \in (0, 0.5] \\ 1 - 2(x-1)^2 & \text{for } x \in (0.5, 1] \\ 1 & \text{for } x > 1. \end{cases}$$

It may by approximated to the ordinary step function by multiplying its argument by a large positive number. The reward is calculated as a sum of basic rewards that play various roles in different stages of learning. Its definition is as follows

$$
\begin{aligned}
r = {} & 0.5(\dot{x}_3 + \dot{x}_4) && (8)\\
& - 0.05 \sum_{i \in I_j} \max\left\{\left|\tau_i^0 + a_i^0\right| - 1, 0\right\} && (9)\\
& - 0.1 \sum_{i \in I_j} \min\{|\tau_i^w|, 50\} && (10)\\
& + \big(f(0.3 \max\{\dot{y}_3 + [y_0 > 0], \dot{y}_4 + [y_9 > 0]\} \\
& \qquad + 0.1) - 1\big)\big(1 - f(1.5(\dot{x}_3 + \dot{x}_4))\big) && (11)\\
& + f(9y_1) - 1 && (12)\\
& + f(5y_2) - 1 && (13)\\
& + f(2y_5) - 1 && (14)
\end{aligned}
$$

for $I_j = \{1, 2, 3, 6, 7, 8\}$. The meaning of the above elements is the following:

- (8) a reward for moving ahead with large as possible speed,
- (9) a penalty for an attempt to apply torch from outside the permissible interval,
- (10) a penalty for the internal force that keeps joint angle in its interval, τ_i^w is a torch applied by the "tendon" to keep i-th joint angle within its bounds (if i-th joint angle is away from both the bounds, then obviously $\tau_i^w = 0$),
- (11) a penalty for not moving the trunk up and keeping paws on the ground when the animal is not moving forward,
- (12) (13) (14) penalties for touching the ground with the heel, the knee, and the head, respectively.

C. The learning algorithm

In order to make Half-Cheetah run, we combine the classic Actor-Critic in the form presented in [3] and the idea of non-Markov periods discussed in this paper. We chose this method because it is known to be very fast although it usually provides suboptimal policies (in contrary to other Actor-Critics like the one presented in [4]). We intend to check to what extent the use of nm-periods is able to introduce any improvement in its work. The algorithm we apply is specified in Table II.

Actions are computed as in Eq. (6) of Sec. IV for $\widetilde{a}$ being an output of a feedforward neural network and ξ being the piece-wise independent autoregressive process with unity variance and $\alpha = 0.8$. The elements of the

6-dimensional action a are transformed into the control stimuli a^0 as

$$a_i^0 = 0.1 a_j$$

where the indexes $i = 1, 2, 3, 6, 7, 8$ correspond to $j = 1, 2, 3, 4, 5, 6$, respectively. The constant 0.1 results from the fact that the control signals a_i^0 cover the interval $[-1, 1]$ while we would prefer the outputs of the network to be of larger scale, e.g. roughly cover the interval $[-10, 10]$.

The second approximator used by the learning algorithm is Critic, i.e. the approximation of the value-function. Critic is also implemented by a feedforward neural network. Both the networks have the form of two layer perceptron with linear output layer. Their hidden layers consist of M^A (Actor) and M^C (Critic) sigmoidal (arctan) elements. Each neuron has a constant input (bias). The initial weights of the hidden layers are drawn randomly from the normal distribution $N(0, 1)$ and the initial weights of the output layers are set to zero.

Actions are applied every 0.02sec. The training consists of a sequence of *trials*. After each step the current trial is artificially stopped with probability $1/250$. Consequently, a trial lasts for a random number of steps with a geometric distribution and, on the average, lasts for 5sec. At the beginning of each trial, the state is reset by putting Half-Cheetah in its initial position 0.1m. above the ground.

The parameters of the algorithm are $M^A = 100$, $M^C = 200$, and $\beta_k^\theta \equiv 2.10^{-6}$, $\beta_k^v \equiv 10^{-5}$. The distribution of ξ is based on the pattern presented in Sec. IV with $\Sigma = I$ and $\alpha = 0.8$. The parameter λ is equal to 0.5. The aggregated reward is defined as a discounted sum of the basic rewards, namely for $\gamma = 0.995$

$$\bar{r}_k = \sum_{i=0}^{l_k - 1} \gamma^i r_{t_k + i}.$$

The discount factor γ_k applied at k-th nm-period is equal to γ^{l_k}. Discounting of the original rewards r_t is thus the same regardless of nm-period lengths. In order to shorten time of simulations, we apply sampling with $\Delta = 4$.

D. Experiments

Figure 3 presents average reward after 10^5 trials of learning for chosen values of l. This parameter defines how long non-Markov periods are. Specifically, $l_k = l$ unless within k-th period the trial is artificially stopped, in which case the period does not take part in learning. Because the objective of learning here is a certain periodic activity, the trials never finish in a terminal state. However, the learning is artificially divided into trials to increase its speed.

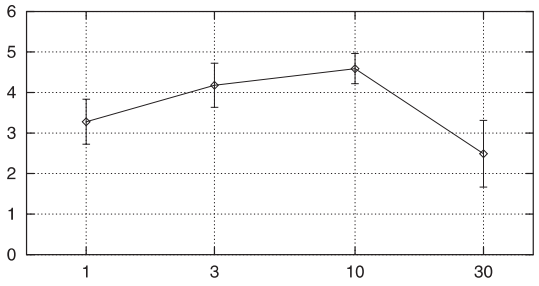

Fig. 3. Average reward vs. length of non-Markov periods.

Notice that for $l = 1$, the algorithm is in fact reduced to the classic Actor-Critic. It seems that the best value of the parameter l is in this case close to 10. It is seen in Fig. 3 that the average performance increases 40% (from 3.28 to 4.59) as l changes from 1 to 10. The question is, whether the policy obtained with $l = 10$ is close to optimal. We can only provide some intuitive assessment in this matter. Namely, for the usual policy obtained for $l = 10$, the behavior of Half-Cheetah resambles quite well the run of a quadrupedal mammal (see Fig. 4). However, for $l = 1$, the behavior is somehow awkward, e.g. the object takes very fine strides or runs on its wrist and heel. For $l = 30$, the learning is unstable or, for smaller step-sizes, very slow; after 10^5 trials it is far from being completed.

VI. Discussion

Our main concern in this work was whether it was possible to optimize a control policy for a complex robotic system using only limited knowledge of its dynamics. Specifically, the knowledge that allowed to build a simulator of these dynamics. Although we did not provide a decisive answer to this question, we succeeded in developing an efficient control policy basing on just a little additional knowledge. Namely, our model of a walking animal learned to run while initially it was only able to stand still. However, this result still seems to be quite optimistic: While it is often relatively easy to design a controller that keeps a system, even a very complex one, close to a certain basic position, design of a controller that governs a nontrivial activity of this system is significantly more demanding.

We also tested here a specific approach to problems that emerge in reinforcement learning of a stationary policy in presence of very fine time discretization. There are reasons to believe that these problems may make learning more difficult or even impossible. The approach we tested is based on dividing the learning process into non-Markov periods. Within this idea, the original problem is transformed to the other one in which the time discretization is reasonably thick. Our experiments confirms that this approach can be successful: while the classic Actor-Critic

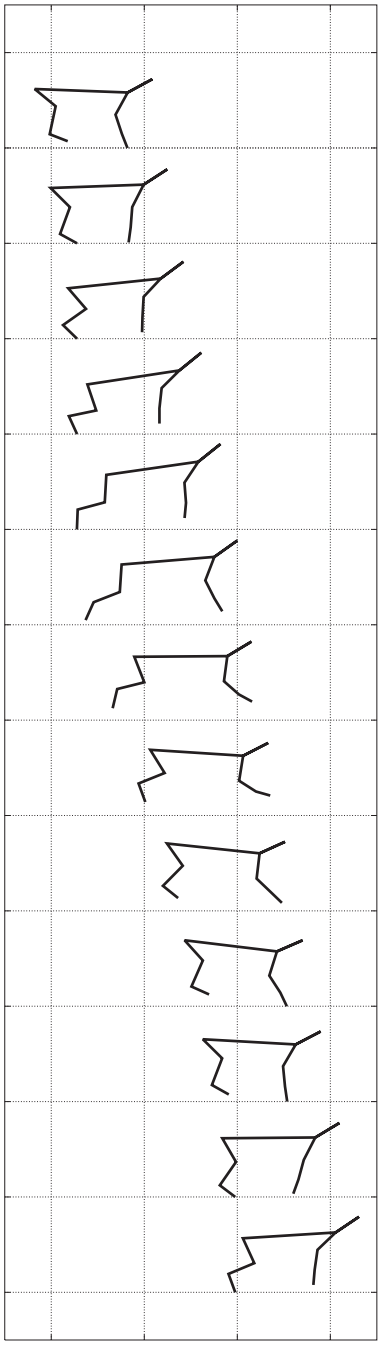

Fig. 4. A typical trajectory of movement of Half-Cheetah for an efficient policy in use.

applied to our test object was obtaining inefficient and awkward control policies, the algorithm based on non-Markov periods developed policies more efficient and conspicuously more deft.

VII. Conclusions

We developed an efficient control policy for a 6-degree-of-freedom walking robot by means of simulated trial and error learning process. The only preliminary knowledge employed to optimize the policy concerned building a simulator of the dynamics of this object and its stabilization in its basic position. The controller of the robot learned to make it run.

We successfully verified our approach to problems that emerge in reinforcement learning of a stationary policy in presence of very fine time discretization. This approach, based on non-Markov periods, significantly outperformed the traditional one that did not remedy the time discretization problems.

References

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike Adaptive Elements That Can Learn Difficult Learning Control Problems. *IEEE Transactions on System Man, and Cybernetics,* vol. SMC-13:834-846, 1983.

[2] C. Watkins and P. Dayan. Q-Learning. *Machine Learning,* vol. 8:279–292, 1992.

[3] H. Kimura and S. Kobayashi. An Analysis of Actor/Critic Algorithm Using Eligibility Traces: reinforcement learning with imperfect value functions, *Proceedings of the ICML-98,* 1998.

[4] V. R. Konda an d J. N. Tsitsiklis. Actor-Critic Algorithms. *SIAM Journal on Control and Optimization,* Vol. 42, No. 4:1143-1166, 2003.

[5] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. *Humanoids2003, 3rd IEEE-RAS International Conference on Humanoid Robots.* Karlsruhe, Germany, Sept.29-30, 2003.

[6] R. Munos. Policy Gradient in Continuous Time. *Journal of Machine Learning Research 7,* pp. 771-791, 2006.

[7] P. Wawrzyński. Reinforcement Learning in Fine Time Discretization. *Lecture Notes in Computer Science, vol. 4431 (Proceedings of ICANNGA 2007),* pp. 470-479, 2007.

[8] R. Williams. Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning,* vol. 8:299–256, 1992.

[9] J. Baxter, P. L. Bartlett, and L. Weaver. Experiments with Infinite-Horizon, Policy-Gradient Estimation, *Journal of Artificial Intelligence Research,* vol. 15:351-381, 2001.

[10] P. Marbach and J. N. Tsitsiklis. Silumation-Based Optimization of Markov Reward Processes. *IEEE Transactions on Automatic Control,* Vol. 46, No. 2, pp. 191-209, 2001.

[11] T. Mori, Y. Nakamura, M. Sato, and S. Ishii. Reinforcement Learning for a CPG-driven Biped Robot, *Nineteenth National Conference on Artificial Intelligence,* pp. 623-630, 2004.

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

[13] M. Buehler, R. Battaglia, A. Cocosco, G. Hawker, J. Sarkis, and K. Yamazaki. Scout: A Simple Quadruped That Walks, Climbs and Runs. *Proceedings of the ICRA1998*, pp. 1707-1712, 1998.

[14] C. M. Chew, J. Pratt, and G. Pratt. Blind Walking of a Planar Bipedal on Sloped Terrain. *Proceedings of the ICRA1999,* pp. 381-386, 1999.

[15] Y. Fukuoka, H. Kimura, and A. H. Cohen. Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain Based on Biological Concepts. *The International Journal of Robotics Research,* vol. 22:187-202, 2003.

[16] D. P. Barnes. Hexapodal Robot Locomotion Over Uneven Terrain. *Proceedings of IEEE Conference on Control Applications,* pp. 441-445, 1998.

[17] A. J. Ijspeert. A Connectionist Central Pattern Generator for the Aquatic and Terrestrial Gaits of a Simulated Salamander. *Biological Cybernetics,* vol. 84:331-348, 2001.

[18] G. Taga, Y. Yamaguchi, and H. Shimizu. Selforganized Control of Bipedal Locomotion by Neural Oscillators in Unpredictable Environment. *Biological Cybernetics,* vol. 65:147-159, 1991.

[19] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement and Control,* 104:205-211, 1982.