Tutorial: Reinforcement Learning for Character Animation

Akihisa Watanabe
akihisa@ruri.waseda.jp
Waseda University
Tokyo, Japan

ABSTRACT

This tutorial presents a detailed exploration of Reinforcement Learning (RL) with a specific focus on its application in physics-based character animation. Centered around Policy Gradient methods, the paper methodically introduces foundational concepts such as Markov Decision Processes and progresses to more advanced topics including Actor-Critic Algorithms and Generalized Advantage Estimation. A key feature of this tutorial is its practical approach, incorporating step-by-step applications and examples drawn from the CS285 course at UC Berkeley. This paper is designed to serve both beginners and experienced individuals in the field, offering a balanced mix of theoretical knowledge and hands-on techniques for applying RL in character animation, and providing insights into the potential benefits and applications of these techniques within the domain. Our code is available at https://github.com/Akihisa-Watanabe/rl-character-animation-tutorial.

CCS CONCEPTS

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

KEYWORDS

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Akihisa Watanabe. 2023. Tutorial: Reinforcement Learning for Character Animation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Developing controllers for physics-based character animation is a significant challenge, merging the need for aesthetic appeal with physical accuracy and reactivity. The integration of reinforcement learning (RL) has transformed this field, enabling the creation of characters that not only move realistically but also exhibit a broad range of athletic abilities, from basic walking to intricate acrobatics. This tutorial delves into these advancements, with a focus on how RL aids in animating characters that interact dynamically with their environments.

We start with an introduction to RL, designed for those new to the concept, then move to its specific application in the realm of character animation. Special attention is given to the insights from Sergey Levine's CS 285 lecture series at UC Berkeley, which provides a fundamental understanding of RL principles and their direct application in computer animation.

The tutorial concludes with a practical guide, featuring step-by-step instructions and examples from current research. It is tailored for readers involved in computer animation, ranging from novices to seasoned professionals. The goal is to impart a thorough understanding of how RL can be utilized to generate advanced, dynamic, and lifelike animations, equipping you with the necessary theoretical knowledge and practical skills.

2 PRELIMINARIES OF REINFORCEMENT LEARNING

Our task is structured as a standard reinforcement learning problem where an agent interacts with its environment according to a policy to maximize a reward signal. In this section, we define basic reinforcement learning concepts, following the standard definition [1, 6]. Reinforcement learning addresses the problem of learning to control a dynamical system in a general sense. The dynamical system is fully defined by a fully-observed or partially-observed Markov decision process (MDP).

Definition 2.1. (Markov decision process). A Markov decision process is defined as a tuple $M = (S, A, T, r, \gamma)$, where S is a set of states $s \in S$, which can be either discrete or continuous (i.e., multi-dimensional vectors), A is a set of actions $a \in A$, which can also be discrete or continuous, T defines a conditional probability distribution of the form $T(s_{t+1}|s_t, a_t)$[1] that describes the dynamics of the system, $r : S \times A \longrightarrow R$ defines a reward function, and $\gamma \in (0, 1]$ is a scalar discount factor.

State: The state $\boldsymbol{s}_t$ consists of a set of features that describe the configuration of the character's body. In the paper [7], the state is defined as follows:

$$\boldsymbol{s}_t = (\boldsymbol{p}_t, \dot{\boldsymbol{p}}_t, \boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \hat{\boldsymbol{p}}_{t+1}, \hat{\boldsymbol{q}}_{t+1}) \tag{1}$$

where

$\boldsymbol{p}_t$:	joint positions in the character's root coordinates
$\dot{\boldsymbol{p}}_t$:	joint linear velocities in the character's root coordinates
$\boldsymbol{q}_t$:	joint rotations in the joints' local coordinates
$\dot{\boldsymbol{q}}_t$:	joint angular velocities in the joints' local coordinates
$\hat{\boldsymbol{p}}_{t+1}$:	target (kinematic) joint positions
$\hat{\boldsymbol{q}}_{t+1}$:	target (kinematic) joint rotations.

[1]We will use $p(s_{t+1}|s_t, a_t)$ for this.

Action: In many prior works[2, 3, 7], each action $\boldsymbol{a}_t$ specifies the target state (such as rotations) for PD controllers positioned at each of the character's joints. For example, we can define action as follows:

$$\boldsymbol{a}_t = \boldsymbol{u}_t \tag{2}$$

where $\boldsymbol{u}_t$ is the target joint angles for the PD controller. At each simulation step, the joint torques τ_t are computed as:

$$\tau_t = \boldsymbol{K}_p \cdot (\boldsymbol{a}_t - \boldsymbol{q}_t^{nr}) - \boldsymbol{K}_d \cdot \dot{\boldsymbol{q}}_t^{nr} \tag{3}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ denote the parameters of the PD controllers that determine the stiffness and damping of each joint, $\boldsymbol{q}_t^{nr}$ and $\dot{\boldsymbol{q}}_t^{nr}$ are the joint rotations and angular velocities of the non-root joints.

Reward: The reward function plays a role in guiding the agent's actions. The reward, denoted as $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$, is computed when the agent performs an action$\boldsymbol{a}_t$ that transitions it from state $\boldsymbol{s}_t$ to state $\boldsymbol{s}_{t+1}$. The design of the reward function can be leveraged to make the agent imitate certain motion data or to specify a particular task that the agent needs to accomplish.

For instance, in the study [7], a part of the reward function used for imitating motion data is given by

$$r_t^v = \exp\left[-\alpha_v \sum_j \left(||\dot{\boldsymbol{q}}_t - \hat{\dot{\boldsymbol{q}}}_t||^2\right)\right]. \tag{4}$$

This reward function measures the discrepancy between the local joint velocities of the simulated motion, $\dot{\boldsymbol{q}}_t$, and the reference motion, $\hat{\dot{\boldsymbol{q}}}_t$.

In another example, the ASE paper by [2] uses a specific reward function to achieve the task of moving a character from its current position, $\boldsymbol{x}_t^{\text{root}}$, to a target point, $\boldsymbol{x}^*$. The reward function used in this context is

$$r_t^G = \exp\left(-0.5||\boldsymbol{x}^* - \boldsymbol{x}_t^{\text{root}}||^2\right) \tag{5}$$

This reward function is designed such that the reward increases as the character gets closer to the target point, thereby encouraging the agent to move towards the target.

The final goal in a reinforcement learning problem is to learn a policy, which defines a distribution over actions conditioned on states, $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. From these definitions, we can derive the trajectory distribution. The trajectory is a sequence of states and actions of length T, given by $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \cdots, \boldsymbol{s}_T, \boldsymbol{a}_T)$, where T may be infinite. The trajectory distribution p_π for a given MDP M and policy π is given by

$$p_\pi(\tau) = p(\boldsymbol{s}_0) \prod_{t=0}^{T} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{6}$$

This represents the probability of obtaining the trajectory τ as a result of the agent selecting actions $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ according to the policy π. Note that because an MDP depends only on information from one time step before, the current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$ capture all the relevant information from the past states and actions, making the future states and actions independent of the past given the present state and action. Therefore, the trajectory distribution is expressed as a product of the policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ and the state transition probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, which simplifies the computation and analysis of the problem.

The objective function in reinforcement learning, denoted as $J(\pi)$, is typically defined as the expected cumulative reward under a given policy π. This can be expressed as an expectation under the trajectory distribution $p_\pi(\tau)$:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]. \tag{7}$$

Here, $\gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ represents the discounted reward at each time step t, where γ is the discount factor that determines the present value of future rewards. The final goal of the agent is to find an optimal policy π^* that maximizes this objective function $J(\pi)$:

$$\pi^* = \underset{\pi}{\text{argmax}}\, J(\pi). \tag{8}$$

In other words, the optimal policy is the one that, on average, results in the highest cumulative reward throughout the trajectory.

3 POLICY GRADIENT METHODS

Policy Gradient Methods are a class of reinforcement learning algorithms that optimize the policy by learning its parameters[4], denoted as θ. This approach is similar to how parameters are updated in traditional supervised learning algorithms. However, in this context, the parameters θ are associated with the policy denoted as π_θ. The parameters are updated iteratively in the direction that maximizes the performance measure $J(\theta)$. The simplest form of this update is given by the gradient ascent rule:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{9}$$

Where, α is the learning rate, and $\nabla_\theta J(\theta)$ is the gradient of the performance measure with respect to the policy parameters.

Theorem 3.1. (Policy Gradient Theorem). Given a policy parameterized by θ, denoted as π_θ, the gradient of the objective function $J(\theta)$ with respect to the policy parameters θ can be expressed as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right)\right]. \tag{10}$$

In this equation, τ represents a trajectory sampled from the policy π_θ, γ^t is the discount factor at time step t, $\nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ is the gradient of the log-probability of the action $\boldsymbol{a}_t$ given state $\boldsymbol{s}_t$ under policy π_θ, and $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the reward received at time step t for taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$.

When $\gamma = 1$, the first term inside the expectation in Equation 10 is exactly the same as the definition of maximum likelihood. Intuitively, this means that we are assigning more weight to more rewarding trajectories by making trajectories with higher rewards more probable. Equivalently, higher-reward trajectories are likely to have more probability to be chosen. In other words, the algorithm is designed to learn from the data in a way that it is most likely to choose trajectories that yield higher rewards.

However, the problem of Policy Gradient Methods is the high variance of the gradient estimates. This is because the gradient is estimated based on sampled trajectories, and the rewards along these trajectories can vary significantly. This variability in rewards leads to a high variance in the gradient estimates, which can in turn lead to unstable learning and make it harder for the algorithm to converge to an optimal policy. Various techniques, such as using a baseline or employing advanced variance reduction methods, have been proposed to address this issue.

3.1 Causality and Baselines

Causality: One simple way to tackle high variance is to use the fact of causality: policy at time t' cannot affect reward at t if $t < t'$. This simple, commonsensical idea allows us to discard some operands in the summation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t | \boldsymbol{s}_t) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right) \right]. \tag{11}$$

and we define the second item in the summation as the "reward-to-go". Notice that in the reward-to-go term, we start the summation from time t instead of 1, by causality. The idea is that we are multiplying the likelihood by smaller numbers due to the reduction of the summation term, so we can reduce the variance to some extent.

Baselines: Another way to reduce variance is to subtract a baseline from the reward function. The baseline $b(s_t)$ helps stabilize learning by reducing variance. The objective function we aim to optimize in policy gradient methods can be represented by the equation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) \times \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) - b(s_t) \right) \right]. \tag{12}$$

The subtraction of the baseline $b(s_t)$ helps in reducing the variance of the estimator, leading to more stable learning.

The baseline $b(s_t)$ takes in two primary inputs: the observations, which encapsulate the states observed by the agent, and the target values, indicative of the returns from the associated states. The training of this critic, or the baseline $b(s_t)$, leverages the mean squared error (MSE) loss. The anticipated output is a loss metric, optimized to update the value function. This loss is calculated as the average least-squares error between the predictions made by the neural network function $b(s_i)$ and the actual target values y_i:

$$l(b) = \mathbb{E}_{s_i, y_i \sim D} \left[||y_i - b(s_i)||^2 \right], \tag{13}$$

where D signifies the dataset amassed by the agent.

3.2 Advantage Function

Now, we introduce the Advantage function, a key concept that further refines policy gradient methods. The Advantage function $A^\pi(s_t, a_t)$ quantifies the relative value of taking a specific action a_t in a given state s_t, compared to the average value of all possible actions in that state according to policy π. The Advantage function is defined as the difference between the Q-function, $Q^\pi(s_t, a_t)$, which represents the expected future return of taking action a_t in state s_t and then following policy π, and the value function, $V^\pi(s_t)$, which represents the expected future return of being in state s_t and then following policy π:

Definition 3.2. The Q-function represents the expected future return of taking action a_t in state s_t and then following policy π for all future timesteps:

$$Q^{\pi,\gamma}(s_t, a_t) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]. \tag{14}$$

Definition 3.3. The value function represents the expected future return of being in state s_t and then following policy π for all future timesteps:

$$V^{\pi,\gamma}(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]. \tag{15}$$

Definition 3.4. Given a policy π and a discount factor γ, the Advantage function $A^{\pi,\gamma}(s_t, a_t)$ quantifies the relative value of taking a specific action a_t in a given state s_t over the average value of all possible actions in that state under the policy π. It is defined as:

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t). \tag{16}$$

The advantage function thus quantifies the relative benefit of taking action a_t in state s_t by comparing the Q-function to the value function. A positive advantage function indicates that taking action a_t in state s_t is expected to yield a higher return than the average return of all possible actions in state s_t. Conversely, a negative advantage function indicates that taking action a_t in state s_t is expected to yield a lower return than the average return of all possible actions in state s_t.

With the advantage function, we can express a policy gradient estimate using a Monte Carlo estimate:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi,\gamma}(s_t, a_t) \right]. \tag{17}$$

This equation can be intuitively understood as follows: a step in the direction of the policy gradient should increase the likelihood of actions that perform better than average, and decrease the likelihood of actions that perform worse than average.

3.3 Actor Critic Algorithm

Building upon the previously introduced concepts of the Advantage function, Q-function, and Value function, we now derive the Actor Critic Algorithm. In the context where s_t, a_t are not random variables, the Q function can be expressed as follows:

$$\begin{aligned} Q^\pi(s_t, a_t) &= r(s_t, a_t) + \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right] \\ &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} \left[V^\pi(s_{t+1}) \right]. \end{aligned} \tag{18}$$

For practical purposes, we introduce a simplifying assumption. Although it may not always hold true in reality, we consider the actual s_{t+1} observed in the current trajectory as a representative of the average of s_{t+1} that we would obtain. This allows us to approximate the Q function as:

$$Q^\pi(s_t, a_t) \approx r(s_t, a_t) + \gamma V^{\pi,\gamma}(s_{t+1}). \tag{19}$$

This approximation is appealing because it simplifies the learning of the advantage function. The advantage function now only depends on V, which in turn only depends on s, unlike Q which depends on both a and s. This reduces the complexity of learning since Q

requires a larger sample size due to its increased dimensionality. Using this approximation, we can substitute into the advantage function:

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t)$$
$$\approx r(s_t, a_t) + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t) \quad (20)$$
$$= \delta_t. \quad (21)$$

This leads us to the actor-critic formulation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau|\pi_\theta)}\left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) \times \left(r(s_t, a_t) + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)\right)\right]. \quad (22)$$

In policy gradient with baselines, there is no bias in our estimation, but there might be high variance due to our single-sample estimation of the Q-function. On the other hand, in the actor-critic algorithm, we have lower variance due to the critic, but we end up having a biased estimation because of the possible modeling error in $V_\phi^{\pi,\gamma}(\cdot)$.

3.4 Generalized Advantage Estimation

The actor-critic advantage function (equation 21) and the Monte Carlo policy gradient (baseline) present a trade-off between bias and variance. The actor-critic advantage function has lower variance but higher bias, while the Monte Carlo policy gradient has lower bias but higher variance. This trade-off becomes more pronounced as we project our trajectory further into the future, where the variance increases due to the insufficiency of the current single sample approximation for future representation. To manage this trade-off, it is advantageous to use the actor-critic based advantage for long-term predictions, which incorporates a larger number of states for better approximation in the long run, and the Monte Carlo based one for short-term predictions, which excels in achieving accurate values in the near term.

To avoid the trajectory extending to a point where the variance becomes excessively large, we can truncate it. This concept is illustrated in Figure 1, where the trajectory is cut off before the variance escalates excessively. We can estimate the advantage function by integrating the two approaches, applying the Monte Carlo approach only for the initial n steps. This is mathematically expressed as:

$$A_n^{\pi,\gamma}(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t) + \gamma^n \hat{V}_\phi^\pi(s_{t+n}). \quad (23)$$

In this equation, we apply an n-step estimator, which sums the reward from the current time to n steps in the future. Often, setting $n > 1$ yields superior performance. The term $\sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t)$ is the Monte Carlo method policy gradient applied over n steps. The last term, $\gamma^n \hat{V}_\phi^\pi(s_{t+n})$, is the discounted value function at the end of the n-step trajectory. This term is used to estimate the future returns beyond the n-step trajectory, and it is where the actor-critic approach comes into play.

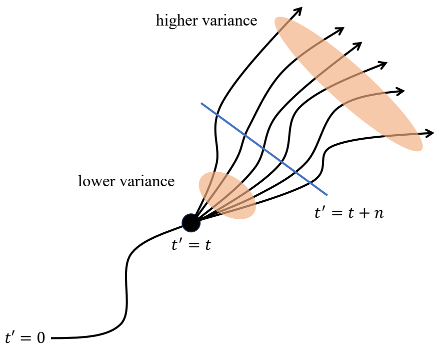

Figure 1: Depiction of trajectory and variance over time. Multiple trajectories are showing that the dispersion is large for distant time steps and small for near time steps. This highlights the need to truncate the trajectory at a certain point ($t' = t+n$) to prevent the variance from escalating excessively.

However, the choice of n is a hyperparameter that needs to be tuned. A small n can lead to high bias and low variance, while a large n can lead to low bias and high variance. To address this issue, we can use a weighted average of all possible n-step estimators, which is known as Generalized Advantage Estimation (GAE) [5]. The GAE is defined as:

$$A_{GAE}^{\pi,\gamma,\lambda}(s_t, a_t) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} A_n^{\pi,\gamma}(s_t, a_t), \quad (24)$$

where $\lambda \in [0, 1]$ is a hyperparameter that determines the trade-off between bias and variance. When $\lambda = 0$, the GAE reduces to the one-step actor-critic estimator, which has high bias and low variance. When $\lambda = 1$, the GAE reduces to the Monte Carlo estimator, which has low bias and high variance. By choosing an appropriate value of λ, we can balance the trade-off between bias and variance.

The equation 24 can be expanded as :

$$A_{GAE}^{\pi,\gamma,\lambda}(s_t, a_t) = (1-\lambda)(A_1^{\pi,\gamma} + \lambda A_2^{\pi,\gamma} + \lambda A_3^{\pi,\gamma} + \cdots)$$
$$= \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \quad (25)$$

For the finite horizon case, the following representation is apt:

$$A_{GAE}^{\pi,\gamma,\lambda}(s_t, a_t) = \sum_{t'=t}^{T-1} (\gamma\lambda)^{t'-t} \delta_{t'}. \quad (26)$$

This formulation allows for an efficient implementation of the generalized advantage estimator, employing a recursive calculation:

$$A_{GAE}^{\pi,\gamma,\lambda}(s_t, a_t) = \delta_t + \gamma\lambda A_{GAE}^{\pi,\gamma,\lambda}(s_{t+1}, a_{t+1}). \quad (27)$$

This approach systematically balances between bias and variance, thus optimizing the learning process.

REFERENCES

[1] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *CoRR* abs/2005.01643 (2020). arXiv:2005.01643 https://arxiv.org/abs/2005.01643

[2] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. 2022. ASE: Large-scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters. *ACM Trans. Graph.* 41, 4, Article 94 (July 2022).

[3] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.* 40, 4, Article 1 (July 2021), 15 pages. https://doi.org/10.1145/3450626.3459670

[4] Jan Peters and Stefan Schaal. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21, 4 (2008), 682–697. https://doi.org/10.1016/j.neunet.2008.02.003 Robotics and Neuroscience.

[5] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[6] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

[7] Haotian Zhang, Ye Yuan, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Xue Bin Peng, and Kayvon Fatahalian. [n. d.]. Learning Physically Simulated Tennis Skills from Broadcast Videos. *ACM Trans. Graph.* ([n. d.]), 14 pages. https://doi.org/10.1145/3592408

Received 20 November 2023; revised 20 November 2023; accepted 20 November 2023