Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?

Xue Bin Peng
University of British Columbia
Vancouver, Canada
xbpeng@cs.ubc.ca

Michiel van de Panne
University of British Columbia
Vancouver, Canada
van@cs.ubc.ca

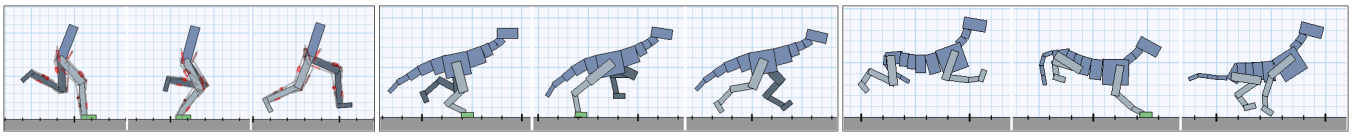

Figure 1: Neural network control policies trained for various simulated planar characters.

ABSTRACT

The use of deep reinforcement learning allows for high-dimensional state descriptors, but little is known about how the choice of action representation impacts learning and the resulting performance. We compare the impact of four different action parameterizations (torques, muscle-activations, target joint angles, and target joint-angle velocities) in terms of learning time, policy robustness, motion quality, and policy query rates. Our results are evaluated on a gait-cycle imitation task for multiple planar articulated figures and multiple gaits. We demonstrate that the local feedback provided by higher-level action parameterizations can significantly impact the learning, robustness, and motion quality of the resulting policies.

CCS CONCEPTS

• **Computing methodologies** → **Animation**; *Physical simulation*; *Control methods*; *Reinforcement learning*;

KEYWORDS

physics-based character animation, motion control, locomotion skills

ACM Reference format:
Xue Bin Peng and Michiel van de Panne. 2017. Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?. In *Proceedings of SCA '17, Los Angeles, CA, USA, July 28-30, 2017,* 13 pages.
DOI: 10.1145/3099564.3099567

1 INTRODUCTION

The introduction of deep learning models to reinforcement learning (RL) has enabled policies to operate directly on high-dimensional, low-level state features. As a result, deep reinforcement learning (DeepRL) has demonstrated impressive capabilities, such as developing control policies that can map from input image pixels to output joint torques [Lillicrap et al. 2015]. However, the motion quality and robustness often falls short of what has been achieved with hand-crafted action abstractions, e.g., Coros et al. [2011]; Geijtenbeek et al. [2013]. While much is known about the learning of state representations, the *choice of action parameterization* is a design decision whose impact is not yet well understood.

Joint torques can be thought of as the most basic and generic representation for driving the movement of articulated figures, given that muscles and other actuation models eventually result in joint torques. However this ignores the intrinsic embodied nature of biological systems, particularly the synergy between control and biomechanics. Passive dynamics, such as elasticity and damping from muscles and tendons, play an integral role in shaping motions: they provide mechanisms for energy storage, and mechanical impedance which generates instantaneous feedback without requiring any explicit computation. Loeb coins the term *preflexes* [Loeb 1995] to describe these effects, and their impact on motion control has been described as providing *intelligence by mechanics* [Blickhan et al. 2007]. This can also be thought of as a kind of partitioning of the computations between the control and physical system.

In this paper we explore the impact of four different actuation models on learning to control dynamic articulated figure locomotion: (1) torques (Tor); (2) activations for musculotendon units (MTU); (3) target joint angles for proportional-derivative controllers (PD); and (4) target joint velocities (Vel). Because Deep RL methods are capable of learning control policies for all these models, it now becomes possible to directly assess how the choice of actuation model affects learning. We also assess the learned policies with respect to robustness, motion quality, and policy query rates. We show that action parameterizations which incorporate local feedback can significantly improve learning speed and performance, while still preserving the generality afforded by torque-level control. Such parameterizations also allow for more complex body structures and subjective improvements in motion quality.

Our specific contributions are: (1) We introduce a DeepRL framework for motion imitation tasks and evaluate the impact of four different actuation models on the learned control policies according

SCA '17, Los Angeles, CA, USA

DOI: 10.1145/3099564.3099567

to four criteria; (2) We propose an optimization approach that combines policy learning and actuator optimization, allowing neural networks to effectively control complex muscle models.

2 RELATED WORK

While significant progress has been made in recent years, motion control of physically simulated characters remains a challenging problem in computer animation. Towards this endeavor, a diverse catalog of controllers have been proposed to tackle various tasks, such as locomotion, manipulation, and other acrobatic skills [Andrews and Kry 2012; Bai et al. 2016; Coros et al. 2009, 2010; Liu et al. 2012]. When constructing controllers, a crucial design decision is the choice of action parameterization, i.e. control signals with which a controller directs the behaviour of a character. Finite state machines (FSM) have been successfully applied to locomotion for human and nonhuman characters [Coros et al. 2008; Peng et al. 2016; Yin et al. 2007]. These models utilize an FSM to organize the phase of a motion into behaviorally similar states, from which a compact set of parameters can be exposed to provide a high-level action abstraction. Examples of action parameters include target joint angles for key joints (e.g. hips and knees), target velocities, virtual forces, end-effector positions, and other task-specific variables. However, crafting FSMs requires significant domain knowledge, often resulting in task-specific controllers. Alternatively, model-based methods leverage access to the equations of motion to directly solve for joint torques via inverse dynamics [de Lasa et al. 2010]. The resulting motions can be parameterized by specifying high-level objectives such as footstep positions and center of mass trajectories [Mordatch et al. 2010]. Biologically-inspired muscle models have also been explored for locomotion [Wang et al. 2012]. However, due to the often high-dimensional nature of muscle models, instead of controlling muscles directly through activations or excitations, previous methods incorporate techniques such as Jacobian transpose control and inverse-dynamics to abstract away the low-level muscle dynamics [Geijtenbeek et al. 2013; Lee et al. 2014]. Other examples of action parameterizations include joint velocities for skilled bicycle stunts [Tan et al. 2014], mixed use of feed-forward torques and joint target angles [Coros et al. 2011], and joint target angles computed by learned linear (time-indexed) feedback strategies [Liu et al. 2016]. Though these models have been successful in reproducing a rich repertoire of motions, the task-specific control structures and assumed knowledge of the dynamics often limit a controller's ability to generalize to new skills and characters, which may entail significant additional engineering.

DeepRL has driven impressive recent advances for motion control, i.e., solving for continuous-action control problems using reinforcement learning. Deep networks have demonstrated promising capacity to directly map high-dimensional state representations to low-level actions such as torques. All four of the action types that we explore have seen previous use in the machine learning literature. WawrzyńSki and Tanwani [2013] use an actor-critic approach with experience replay to learn skills for an octopus arm (actuated by a simple muscle model) and a planar half cheetah (actuated by joint-based PD-controllers). Recent work on deterministic policy gradients [Lillicrap et al. 2015] and on RL benchmarks, e.g., OpenAI Gym, generally use joint torques as the action space, as do the test suites in recent work [Schulman et al. 2015] on using generalized advantage estimation. Other recent work uses: the PR2 effort control interface as a proxy for torque control [Levine et al. 2015]; joint velocities [Gu et al. 2016]; velocities under an implicit control policy [Mordatch et al. 2015]; or provide abstract actions [Hausknecht and Stone 2015]. Our learning procedures are based on prior work using actor-critic approaches with positive temporal difference updates [Van Hasselt 2012].

Work in biomechanics has long recognized the embodied nature of the control problem and the view that musculotendon systems provide *"preflexes"* [Loeb 1995] that effectively provide a form intelligence by mechanics [Blickhan et al. 2007; van Soest and Bobbert 1993], as well as allowing for energy storage. Control methods in robotics use a mix of actuation types, including direct-drive torques (or their virtualized equivalents), series elastic actuators, PD control, and velocity control. These methods often rely heavily on model-based solutions and thus we do not describe these in further detail here.

3 BACKGROUND

Our task will be structured as a standard reinforcement learning problem where an agent interacts with its environment according to a policy in order to maximize a reward signal. The policy $\pi(s,a) = p(a|s)$ represents the conditional probability density function of selecting action $a \in A$ in state $s \in S$. At each control step t, the agent observes a state s_t and samples an action a_t from π. The environment in turn responds with a scalar reward r_t, and a new state $s'_t = s_{t+1}$ sampled from its dynamics $p(s'|s,a)$. For a parameterized policy $\pi_\theta(s,a)$, the goal of the agent is learn the parameters θ which maximizes the expected cumulative reward

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \middle| \pi_\theta\right]$$

with $\gamma \in [0,1]$ as the discount factor, and T as the horizon. The gradient of the expected reward $\nabla_\theta J(\pi_\theta)$ can be determined according to the policy gradient theorem [Sutton et al. 2001], which provides a direction of improvement to adjust the policy parameters θ.

$$\nabla_\theta J(\pi_\theta) = \int_S d_\theta(s) \int_A \nabla_\theta \log(\pi_\theta(s,a)) \mathbb{A}(s,a) da\, ds$$

where $d_\theta(s) = \int_S \sum_{t=0}^{T} \gamma^t p_0(s_0) p(s_0 \rightarrow s|t,\pi_\theta) ds_0$ is the discounted state distribution, $p_0(s)$ represents the initial state distribution, and $p(s_0 \rightarrow s|t,\pi_\theta)$ models the likelihood of reaching state s by starting at s_0 and following the policy $\pi_\theta(s,a)$ for T steps [Silver et al. 2014]. $\mathbb{A}(s,a)$ represents a generalized advantage function. The choice of advantage function gives rise to a family of policy gradient algorithms, but in this work, we will focus on the one-step temporal difference advantage function [Schulman et al. 2015]

$$\mathbb{A}(s_t,a_t) = r_t + \gamma V(s'_t) - V(s_t)$$

where $V(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \middle| s_0 = s, \pi_\theta\right]$ is the state-value function, and can be defined recursively via the Bellman equation

$$V(s_t) = \mathop{\mathbb{E}}_{r_t, s'_t}\left[r_t + \gamma V(s'_t) \middle| s_t, \pi_\theta\right]$$

A parameterized value function $V_\phi(s)$, with parameters ϕ, can be learned iteratively in a manner similar to Q-Learning by minimizing the Bellman loss [Mnih et al. 2015],

$$L(\phi) = \mathop{\mathbb{E}}_{s_t, r_t, s_t'} \left[\frac{1}{2} \left(y_t - V_\phi(s_t) \right)^2 \right], \qquad y_t = r_t + \gamma V_\phi(s_t')$$

π_θ and V_ϕ can be trained in tandem using an actor-critic framework [Konda and Tsitsiklis 2000].

In this work, each policy will be represented as a Gaussian distribution with a parameterized mean $\mu_\theta(s)$ and fixed covariance matrix $\Sigma = \text{diag}\{\sigma_i^2\}$, where σ_i is manually specified for each action parameter. Actions can be sampled from the distribution by applying Gaussian noise to the mean action

$$a_t = \mu_\theta(s_t) + \mathcal{N}(0, \Sigma)$$

The corresponding policy gradient will assume the form

$$\nabla_\theta J(\pi_\theta) = \int_S d_\theta(s) \int_A \nabla_\theta \mu_\theta(s) \Sigma^{-1} \left(a - \mu_\theta(s) \right) \mathbb{A}(s, a) da\, ds$$

which can be interpreted as shifting the mean of the action distribution towards actions that lead to higher than expected rewards, while moving away from actions that lead to lower than expected rewards.

4 TASK REPRESENTATION

4.1 Reference Motion

In our task, the goal of a policy is to imitate a given reference motion $\{q_t^*\}$ which consists of a sequence of kinematic poses q_t^* in reduced coordinates. The reference velocity $\dot{q}_t^*$ at a given time t is approximated by finite-difference $\dot{q}_t^* \approx \frac{q_{t+\triangle t}^* - q_t^*}{\triangle t}$. Reference motions are generated via either using a recorded simulation result from a preexisting controller ("Sim"), or via manually-authored keyframes. Since hand-crafted reference motions may not be physically realizable, the goal is to closely reproduce a desired motion while satisfying physical constraints.

4.2 States

To define the state of the agent, a feature transformation $\Phi(q, \dot{q})$ is used to extract a set of features from the reduced-coordinate pose q and velocity $\dot{q}$. The features consist of the height of the root (pelvis) from the ground, the position of each link with respect to the root, and the center of mass velocity of each link. When training a policy to imitate a cyclic reference motion $\{q_t^*\}$, knowledge of the motion phase can help simplify learning. Therefore, we augment the state features with a set of target features $\Phi(q_t^*, \dot{q_t}^*)$, resulting in a combined state represented by $s_t = (\Phi(q_t, \dot{q_t}), \Phi(q_t^*, \dot{q_t}^*))$. Similar results can also be achieved by providing a single motion phase variable as a state feature, as we show in Figure 19 (supplemental material).

4.3 Actions

We train separate policies for each of the four actuation models, as described below. Each actuation model also has related actuation parameters, such as feedback gains for PD-controllers and musculotendon properties for MTUs. These parameters can be manually specified, as we do for the PD and Vel models, or they can be optimized for the task at hand, as for the MTU models. Table 1 provides a list of actuation parameters for each actuation model.

Target Joint Angles (PD):. Each action represents a set of target angles $\hat{q}$, where $\hat{q}^i$ specifies the target angles for joint i. $\hat{q}$ is applied to PD-controllers which compute torques according to

$$\tau^i = k_p^i(\hat{q}^i - q^i) + k_d^i(\hat{\dot{q}}^i - \dot{q}^i)$$

where $\hat{\dot{q}}^i = 0$, and k_p^i and k_d^i are manually-specified gains.

Target Joint Velocities (Vel): Each action specifies a set of target velocities $\hat{\dot{q}}$ which are used to compute torques according to

$$\tau^i = k_d^i(\hat{\dot{q}}^i - \dot{q}^i)$$

where the gains k_d^i are specified to be the same as those used for target angles.

Torques (Tor): Each action directly specifies torques for every joint, and constant torques are applied for the duration of a control step. Due to torque limits, actions are bounded by manually specified limits for each joint. Unlike the other actuation models, the torque model does not require additional actuation parameters, and can thus be regarded as requiring the least amount of domain knowledge. Torque limits are excluded from the actuation parameter set as they are common for all parameterizations.

Muscle Activations (MTU):. Each action specifies activations for a set of musculotendon units (MTU). Detailed modeling and implementation information are available in Wang et al. [2012]. Each MTU is modeled as a contractile element (CE) attached to a serial elastic element (SE) and parallel elastic element (PE). The force exerted by the MTU can be calculated according to

$$F_{MTU} = F_{SE} = F_{CE} + F_{PE}$$

Both F_{SE} and F_{PE} are modeled as passive springs, while F_{CE} is actively controlled according to

$$F_{CE} = a_{MTU} F_0 f_l(l_{CE}) f_v(v_{CE})$$

with a_{MTU} being the muscle activation, F_0 the maximum isometric force, l_{CE} and v_{CE} being the length and velocity of the contractile element. The functions $f_l(l_{CE})$ and $f_v(v_{CE})$ represent the force-length and force-velocity relationships, modeling the variations in the maximum force that can be exerted by a muscle as a function of its length and contraction velocity. Analytic forms are available in Geyer et al. [2003]. Activations are bounded between [0, 1]. The length of each contractile element l_{CE} are included as state features. To simplify control and reduce the number of internal state parameters per MTU, the policies directly control muscle activations instead of indirectly through excitations [Wang et al. 2012].

4.4 Reward

The reward function consists of a weighted sum of terms that encourage the policy to track a reference motion.

$$r = w_{pose} r_{pose} + w_{vel} r_{vel} + w_{end} r_{end} + w_{root} r_{root} + w_{com} r_{com}$$

$$w_{pose} = 0.5,\ w_{vel} = 0.05,\ w_{end} = 0.15,\ w_{root} = 0.1,\ w_{com} = 0.2$$

Actuation Model	Actuation Parameters
Target Joint Angles (PD)	proportional gains k_p, derivative gains k_d
Target Joint Velocities (Vel)	derivative gains k_d
Torques (Tor)	none
Muscle Activations (MTU)	optimal contractile element length, serial elastic element rest length, max isometric force, pennation, moment arm, max moment arm orientation, rest joint orientation

Table 1: Actuation models and their respective actuation parameters.

Details of each term are available in the supplemental material. r_{pose} penalizes deviation of the character pose from the reference pose, and r_{vel} penalizes deviation of the joint velocities. r_{end} and r_{root} accounts for the position error of the end-effectors and root. r_{com} penalizes deviations in the center of mass velocity from that of the reference motion.

4.5 Initial State Distribution

We design the initial state distribution, $p_0(s)$, to sample states uniformly along the reference trajectory. At the start of each episode, q^* and $\dot{q}^*$ are sampled from the reference trajectory, and used to initialize the pose and velocity of the agent. This helps guide the agent to explore states near the target trajectory. Figure 2 illustrates a comparison between fixed and sampled initial state distributions.

5 ACTOR-CRITIC LEARNING ALGORITHM

Instead of directly using the temporal difference advantage function, we adapt a positive temporal difference (PTD) update as proposed by Van Hasselt [2012].

$$\mathbb{A}(s,a) = I\left[\delta > 0\right] = \begin{cases} 1, & \delta > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\delta = r + \gamma V(s') - V(s)$$

Unlike more conventional policy gradient methods, PTD is less sensitive to the scale of the advantage function and avoids instabilities that can result from negative TD updates. For a Gaussian policy, a negative TD update moves the mean of the distribution away from an observed action, effectively shifting the mean towards an unknown action that may be no better than the current mean action [Van Hasselt 2012]. In expectation, these updates converges to the true policy gradient, but for stochastic estimates of the policy gradient, these updates can cause the agent to adopt undesirable behaviours which affect subsequent experiences collected by the agent. Furthermore, we incorporate experience replay, which has been demonstrated to improve stability when training neural network policies with Q-learning in discrete action spaces [Mnih et al. 2015]. Experience replay often requires off-policy methods, such as importance weighting, to account for differences between the policy being trained and the behavior policy used to generate experiences [WawrzyńSki and Tanwani 2013]. However, we have not found importance weighting to be beneficial for PTD.

Stochastic policies are used during training for exploration, while deterministic policies are used for evaluation at runtime. The choice between a stochastic and deterministic policy can be specified by the addition of a binary indicator variable $\lambda \in [0, 1]$

$$a_t = \mu_\theta(s_t) + \lambda\mathcal{N}(0, \Sigma)$$

where $\lambda = 1$ corresponds to a stochastic policy with exploration noise, and $\lambda = 0$ corresponds to a deterministic policy that always selects the mean of the distribution. Noise from a stochastic policy will result in a state distribution that differs from that of the deterministic policy at runtime. To mitigate this discrepancy, we incorporate ϵ-greedy exploration to the original Gaussian exploration strategy. During training, λ is determined by a Bernoulli random variable $\lambda \sim \text{Ber}(\epsilon)$, where $\lambda = 1$ with probability $\epsilon \in [0, 1]$. The exploration rate ϵ is annealed linearly from 1 to 0.2 over 500k iterations, which slowly adjusts the state distribution encountered during training to better resemble the distribution at runtime. Since the policy gradient is defined for stochastic policies, only tuples recorded with exploration noise (i.e. $\lambda = 1$) can be used to update the actor, while the critic can be updated using all tuples.

Training proceeds episodically, where the initial state of each episode is sampled from $p_0(s)$, and the episode duration is drawn from an exponential distribution with a mean of 2s. To discourage falling, an episode will also terminate if any part of the character's trunk makes contact with the ground for an extended period of time, leaving the agent with zero reward for all subsequent steps. Algorithm 1 summarizes the complete learning process.

MTU Actuator Optimization: Actuation models such as MTUs are defined by further parameters whose values impact performance [Geijtenbeek et al. 2013]. Geyer et al. [2003] uses existing anatomical estimates for humans to determine MTU parameters, but such data may not be readily available for more arbitrary creatures. Alternatively, Geijtenbeek et al. [2013] uses covariance matrix adaptation (CMA), a derivative-free evolutionary search strategy, to simultaneously optimize MTU and policy parameters. This approach is limited to policies with reasonably low dimensional parameter spaces, and is thus ill-suited for neural network models

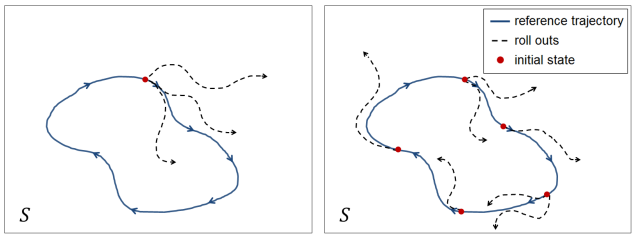

Figure 2: Left: fixed initial state biases agent to regions of the state space near the initial state, particular during early iterations of training. Right: initial states sampled from reference trajectory allows agent to explore state space more uniformly around the reference trajectory.

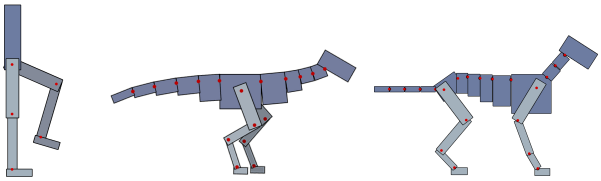

Figure 3: Simulated articulated figures and their state representation. Revolute joints connect all links. left-to-right: 7-link biped; 19-link raptor; 21-link dog.

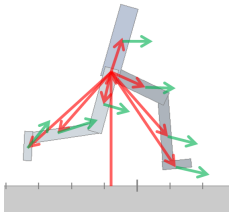

Figure 4: State features: root height, relative position (red) of each link with respect to the root and their respective linear velocity (green).

with hundreds of thousands of parameters. To avoid manual-tuning of actuation parameters, we propose a heuristic approach that alternates between policy learning and actuator optimization.

The actuation parameters ψ can be interpreted as a parameterization of the dynamics of the system $p(s'|s,a,\psi)$. The expected cumulative reward can then be re-parameterized according to

$$J(\pi_\theta,\psi)=\int_S d_\theta(s|\psi)\int_A \pi_\theta(s,a)\mathbb{A}(s,a)da\,ds$$

where $d_\theta(s|\psi)=\int_S\sum_{t=0}^{T}\gamma^t p_0(s_0)p(s_0\rightarrow s|t,\pi_\theta,\psi)ds_0$. θ and ψ are then learned in an alternating fashion as per Algorithm 2. This alternating method optimizes both the control and dynamics in order to maximize the expected value of the agent, as analogous to the role of evolution in biomechanics. During each pass, the policy parameters θ are trained to improve the agent's expected value for a fixed set of actuation parameters ψ. Next, ψ is optimized using CMA to improve performance while keeping θ fixed. The expected value of each CMA sample of ψ is estimated using the average undiscounted cumulative reward over multiple rollouts.

6 RESULTS

The motions are best seen in the supplemental video. We evaluate the action parameterizations by training policies for a simulated 2D biped, dog, and raptor as shown in Figure 3. Depending on the agent and the actuation model, our systems have 58–214 state dimensions, 6–44 action dimensions, and 0–282 actuation parameters, as summarized in Table 3 (supplemental materials). The MTU models have at least double the number of action parameters because they come in antagonistic pairs. As well, additional MTUs are used for the legs to more accurately reflect bipedal biomechanics. This includes MTUs that span multiple joints.

Algorithm 1 Actor-critic Learning Using Positive Temporal Differences

$\theta \leftarrow$ random weights
$\phi \leftarrow$ random weights

while not done **do**
  for $step = 1,...,m$ **do**
    $s \leftarrow$ start state
    $\lambda \leftarrow \text{Ber}(\epsilon_t)$
    $a \leftarrow \mu_\theta(s)+\lambda\mathcal{N}(0,\Sigma)$
    Apply a and simulate forward 1 step
    $s' \leftarrow$ end state
    $r \leftarrow$ reward
    $\tau \leftarrow (s,a,r,s',\lambda)$
    store τ in replay memory

    if episode terminated **then**
      Sample s_0 from $p_0(s)$
      Reinitialize state s to s_0
    end if
  end for

  Update critic:
  Sample minibatch of n tuples $\{\tau_i=(s_i,a_i,r_i,\lambda_i,s_i')\}$ from replay memory
  for each τ_i **do**
    $\delta_i \leftarrow r_i+\gamma V_\phi(s_i')-V_\phi(s_i)$
    $\phi \leftarrow \phi+\alpha_V\frac{1}{n}\delta_i\nabla_\phi V_\phi(s_i)$
  end for

  Update actor:
  Sample minibatch of n tuples $\{\tau_j=(s_j,a_j,r_j,\lambda_j,s_j')\}$ from replay memory where $\lambda_j=1$
  for each τ_j **do**
    $\delta_j \leftarrow r_j+\gamma V_\phi(s_j')-V_\phi(s_j)$
    if $\delta_j>0$ **then**
      $\nabla a_j \leftarrow a_j-\mu_\theta(s_j)$
      $\nabla\tilde{a}_j \leftarrow \text{BoundActionGradient}(\nabla a_j,\mu_\theta(s_j))$
      $\theta \leftarrow \theta+\alpha_\pi\frac{1}{n}\nabla_\theta\mu_\theta(s_j)\Sigma^{-1}\nabla\tilde{a}_j$
    end if
  end for
end while

Algorithm 2 Alternating Actuator Optimization

$\theta \leftarrow \theta_0$
$\psi \leftarrow \psi_0$
while not done **do**
  $\theta \leftarrow \text{argmax}_{\theta'} J(\pi_{\theta'},\psi)$ with Algorithm 1
  $\psi \leftarrow \text{argmax}_{\psi'} J(\pi_\theta,\psi')$ with CMA
end while

Each policy is represented by a three layer neural network, as illustrated in Figure 5 with 512 and 256 fully-connected units, followed by a linear output layer where the number of output units

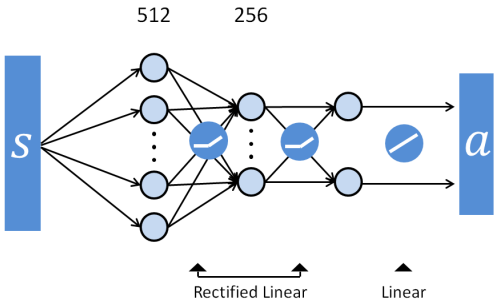

Figure 5: Neural network architecture. Each policy is represented by a three layered network, with 512 and 256 fully-connected hidden units, followed by a linear output layer.

vary according to the number of action parameters for each character and actuation model. ReLU activation functions are used for both hidden layers. Each network has approximately 200k parameters. The value function is represented by a similar network, except having a single linear output unit. The policies are queried at 60Hz for a control step of about 0.0167s. Each network is randomly initialized and trained for about 1 million iterations, requiring 32 million tuples, the equivalent of approximately 6 days of simulated time. Each policy requires about 10 hours for the biped, and 20 hours for the raptor and dog on an 8-core Intel Xeon E5-2687W.

Only the actuation parameters for MTUs are optimized with Algorithm 2, since the parameters for the other actuation models are few and reasonably intuitive to determine. The initial actuation parameters ψ_0 are manually specified, while the initial policy parameters θ_0 are randomly initialized. Each pass optimizes ψ using CMA for 250 generations with 16 samples per generation, and θ is trained for 250k iterations. Parameters are initialized with values from the previous pass. The expected value of each CMA sample of ψ is estimated using the average cumulative reward over 16 rollouts with a duration of 10s each. Separate MTU parameters are optimized for each character and motion. Each set of parameters is optimized for 6 passes following Algorithm 2, requiring approximately 50 hours. Figure 9 illustrates the performance improvement per pass. Figure 10 compares the performance of MTUs before and after optimization. For most examples, the optimized actuation parameters significantly improve learning speed and final performance. For the sake of comparison, after a set of actuation parameters has been optimized, a new policy is retrained with the new set of actuation parameters and its performance compared to the other actuation models.

Policy Performance and Learning Speed: Figure 6 shows learning curves for the policies and the performance of the final policies are summarized in Table 4 (supplemental material). Performance is evaluated using the normalized cumulative reward (NCR), calculated from the average cumulative reward over 32 episodes with lengths of 10s, and normalized by the maximum and minimum cumulative reward possible for each episode. No discounting is applied when calculating the NCR. The initial state of each episode is sampled from the reference motion according to $p(s_0)$. To compare learning speeds, we use the normalized area under each learning

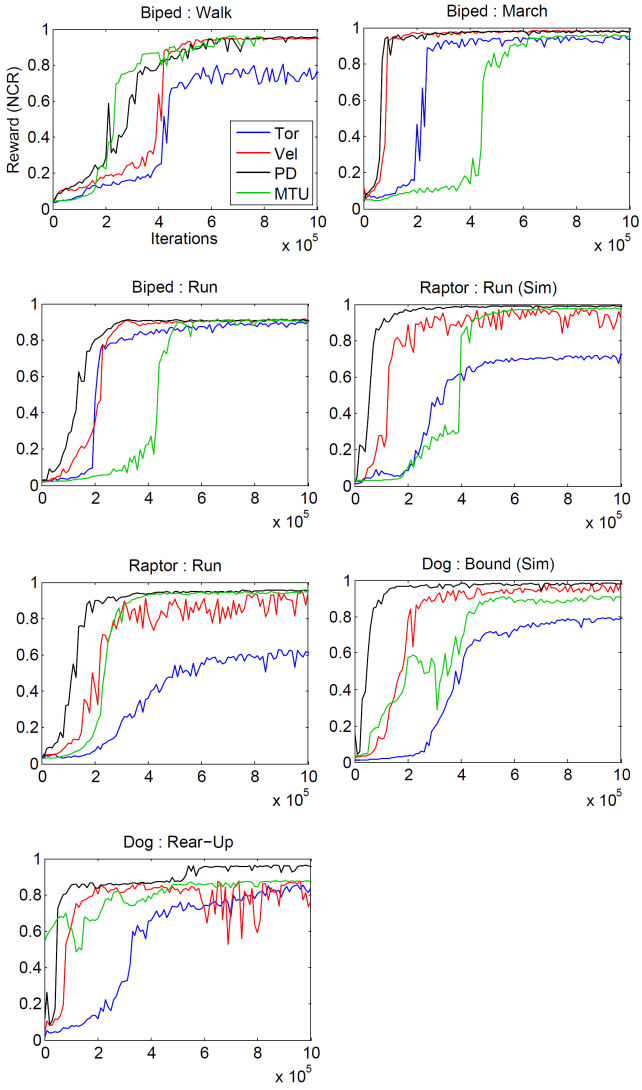

Figure 6: Learning curves for each policy during 1 million iterations.

curve (AUC) as a proxy for the learning speed of a particular actuation model, where 0 represents the worst possible performance and no progress during training, and 1 represents the best possible performance without requiring training.

PD performs well across all examples, achieving comparable-to-the-best performance for all motions. PD also learns faster than the other parameterizations for 5 of the 7 motions. The final performance of Tor is among the poorest for all the motions. Differences in performance appear more pronounced as characters become more complex. For the simple 7-link biped, most parameterizations achieve similar performance. However, for the more complex dog and raptor, the performance of Tor policies deteriorate with respect to other policies such as PD and Vel. MTU policies often exhibited the slowest learning speed, which may be a consequence of the higher dimensional action spaces, i.e., requiring antagonistic muscle pairs, and complex muscle dynamics. Nonetheless, once

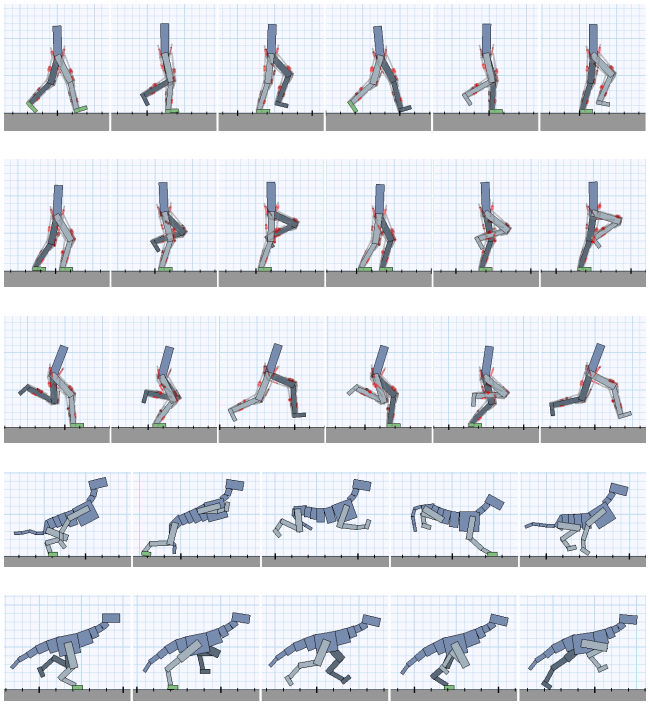

Figure 7: Simulated motions. The biped uses an MTU action space while the dog and raptor are driven by a PD action space.

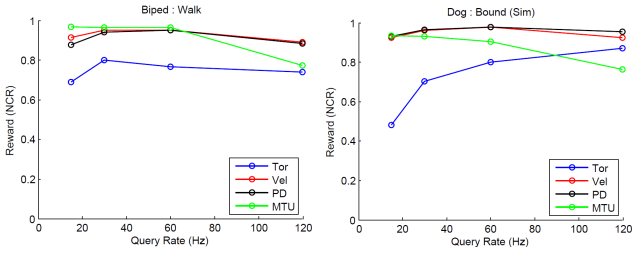

Figure 8: Performance of policies with different query rates for the biped (left) and dog (right). Separate policies are trained for each query rate.

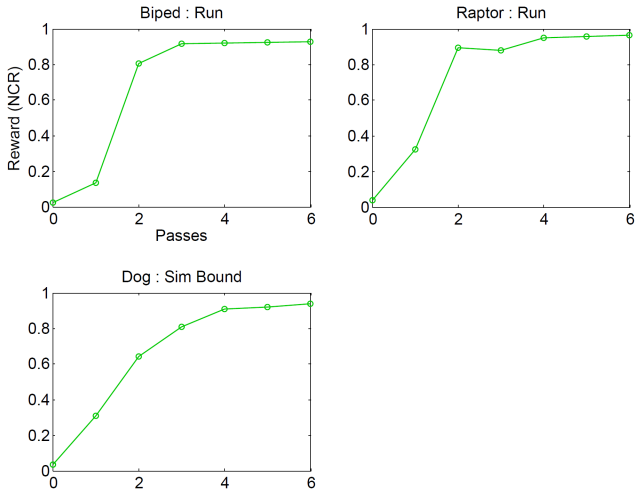

Figure 9: Performance of intermediate MTU policies and actuation parameters per pass of actuator optimization following Algorithm 2.

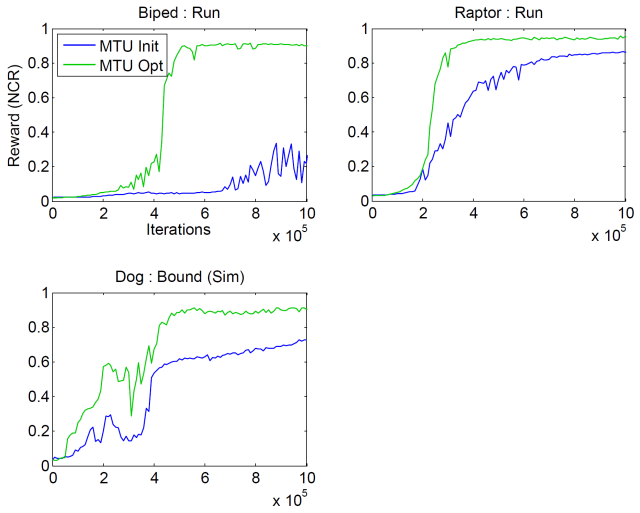

Figure 10: Learning curves comparing initial and optimized MTU parameters.

optimized, the MTU policies produce more natural motions and responsive behaviors as compared to other parameterizations. We note that the naturalness of motions is not well captured by the reward, since it primarily gauges similarity to the reference motion, which may not be representative of natural responses when perturbed from the nominal trajectory. A sensitivity analysis of the policies' performance to variations in network architecture and hyperparameters is available in the supplemental material.

MTU Actuator Optimization: Figure 9 illustrates the improvement in performance during the MTU actuator optimization process, as applied to motions for three different agents. Figure 10 compares the learning curves for the initial and final MTU parameters, for the same three motions. The policies trained with the optimized MTU parameters learn significantly faster than the initial parameter set and also achieve better final performance.

Policy Robustness: To evaluate robustness, we record the NCR achieved by each policy when subjected to external perturbations. The perturbations assume the form of random forces applied to the trunk of the characters. Figure 11 illustrates the performance of the policies when subjected to perturbations of different magnitudes. The magnitude of the forces are constant, but the direction varies randomly. Each force is applied for 0.1 to 0.4s, with 1 to 4s between each perturbation. Performance is estimated using the average over

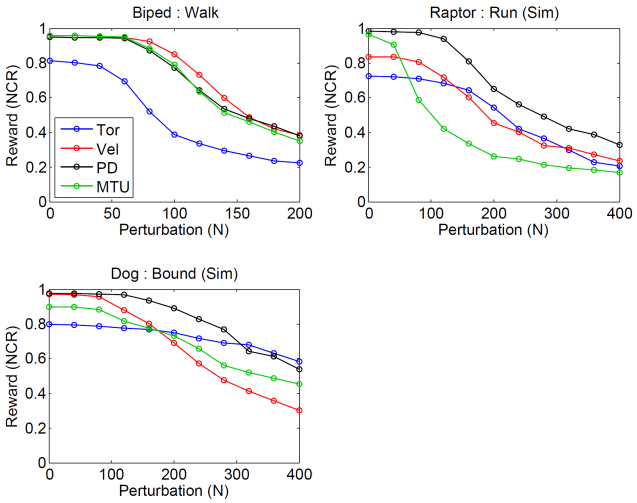

Figure 11: Performance when subjected to random perturbation forces of different magnitudes.

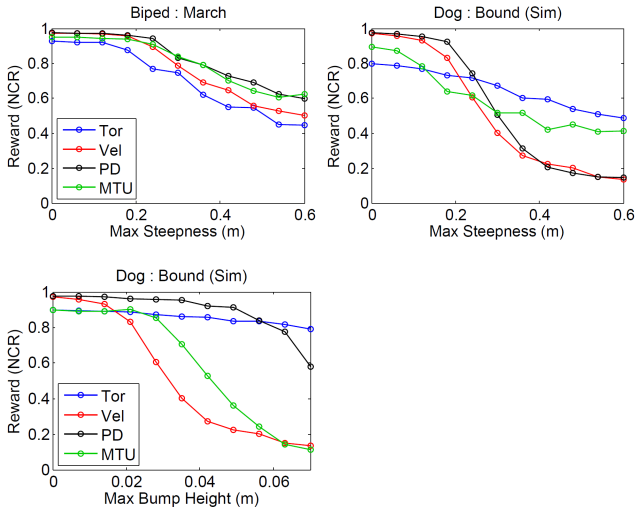

Figure 12: Performance of action parameterizations when traveling across randomly generated irregular terrain. (top) Dog and biped traveling across randomly generated slopes with bounded maximum steepness. (bottom) Dog running across bumpy terrain, where the height of each bump varies uniformly between 0 and a specified maximum height.

128 episodes of length 20s each. For the biped walk, the Tor policy is significantly less robust than those for the other types of actions, while the MTU policy is the least robust for the raptor run. Overall, the PD policies are among the most robust for all the motions. In addition to external forces, we also evaluate the robustness when locomoting over randomly generated terrain consisting of bumps with varying heights and slopes with varying steepness. We evaluate the performance on irregular terrain (Figure 12). There are a few consistent patterns for this test. The Vel and MTU policies are significantly worse than the Tor and PD policies for the dog bound on the bumpy terrain. The unnatural jittery behavior of the dog Tor policy proves to be surprisingly robust for this scenario. We suspect that the behavior prevents the trunk from contacting the ground for extended periods for time, and thereby escaping our system's fall detection.

Query Rate: Figure 8 compares the performance of different parameterizations for different policy query rates. Separate policies are trained with queries of 15Hz, 30Hz, 60Hz, and 120Hz. Actuation models that incorporate low-level feedback such as PD and Vel, appear to cope more effectively to lower query rates, while the Tor degrades more rapidly at lower query rates. It is not yet obvious to us why MTU policies appear to perform better at lower query rates and worse at higher rates. Lastly, Figure 18 (supplemental material) shows the policy outputs as a function of time for the four actuation models, for a particular joint, as well as showing the resulting joint torque. Interestingly, the MTU action is visibly smoother than the other actions and results in joint torques profiles that are smoother than those seen for PD and Vel.

Additional Experiments: We present a number of additional experiments in the supplemental material. These include an analysis of the sensitivity learning results with respect to different random intializations, different choices of PD gains, different amounts of exploration noise, and reference motions that were originally synthesized via alternative means, e.g., MTU-based simulations.

7 CONCLUSIONS

Our experiments suggest that action parameterizations that include basic local feedback, such as PD target angles, MTU activations, or target velocities, can improve policy performance and learning speed across different motions and character morphologies. Such models more accurately reflect the embodied nature of control in biomechanical systems, and the role of mechanical components in shaping the overall dynamics of motions and their control. The difference between low-level and high-level action parameterizations grow with the complexity of the characters, with high-level parameterizations scaling more gracefully to complex characters. As a caveat, there may well be tasks, such as impedance control, where lower-level action parameterizations such as Tor may prove advantageous. We believe that no single action parameterization will be the best for all problems. However, since objectives for motion control problems are often naturally expressed in terms of kinematic properties, higher-level actions such as target joint angles and velocities may be effective for a wide variety of motion control problems. We hope that our work will help open discussions around the choice of action parameterizations.

Our results have only been demonstrated on planar articulated figure simulations; the extension to 3D currently remains as future work. Furthermore, our current torque limits are still large as compared to what might be physically realizable. Tuning actuation parameters for complex actuation models such as MTUs remains challenging. Though our actuator optimization technique is able to improve performance as compared to manual tuning, the resulting parameters may still not be optimal for the desired task. Therefore, our comparisons of MTUs to other action parameterizations may not be reflective of the full potential of MTUs with more optimal

actuation parameters. Furthermore, our actuator optimization currently tunes parameters for a specific motion, rather than a larger suite of motions, as might be expected in nature.

Since the reward terms are mainly expressed in terms of joint positions and velocities, it may seem that it is inherently biased in favour of PD and Vel. However, the real challenges for the control policies lie elsewhere, such as learning to compensate for gravity and ground-reaction forces, and learning foot-placement strategies that are needed to maintain balance for the locomotion gaits. The reference pose terms provide little information on how to achieve these hidden aspects of motion control that will ultimately determine the success of the locomotion policy. While we have yet to provide a concrete answer for the generalization of our results to different reward functions, we believe that the choice of action parameterization is a design decision that deserves greater attention regardless of the choice of reward function.

Finally, it is reasonable to expect that evolutionary processes would result in the effective co-design of actuation mechanics and control capabilities. Developing optimization and learning algorithms to allow for this kind of co-design is a fascinating possibility for future work.

REFERENCES

Sheldon Andrews and Paul G. Kry. 2012. Policies for Goal Directed Multi-Finger Manipulation. In *Workshop on Virtual Reality Interaction and Physical Simulation*, Jan Bender, Arjan Kuijper, Dieter W. Fellner, and Eric Guerin (Eds.). The Eurographics Association. DOI: http://dx.doi.org/10.2312/PE/vriphys/vriphys12/137-145

Yunfei Bai, Wenhao Yu, and C Karen Liu. 2016. Dexterous manipulation of cloth. In *Computer Graphics Forum*, Vol. 35.

Reinhard Blickhan, Andre Seyfarth, Hartmut Geyer, Sten Grimmer, Heiko Wagner, and Michael Günther. 2007. Intelligence by mechanics. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 365, 1850 (2007), 199–220.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust Task-based Control Policies for Physics-based Characters. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 28, 5 (2009), Article 170.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Transctions on Graphics* 29, 4 (2010), Article 130.

Stelian Coros, Philippe Beaudoin, KangKang Yin, and Michiel van de Panne. 2008. Synthesis of Constrained Walking Skills. *ACM Trans. Graph. (Proc. Siggraph Asia)* 27, 5 (2008).

Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. 2011. Locomotion Skills for Simulated Quadrupeds. *ACM Transactions on Graphics* 30, 4 (2011), Article TBD.

Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. 2010. Feature-based locomotion controllers. In *ACM Transactions on Graphics (TOG)*, Vol. 29. ACM, 131.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics* 32, 6 (2013).

Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. 2003. Positive force feedback in bouncing gaits? *Proc. Royal Society of London B: Biological Sciences* 270, 1529 (2003), 2173–2183.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2016. Deep Reinforcement Learning for Robotic Manipulation. *arXiv preprint arXiv:1610.00633* (2016).

Matthew J. Hausknecht and Peter Stone. 2015. Deep Reinforcement Learning in Parameterized Action Space. *CoRR* abs/1511.04143 (2015).

Vijay Konda and John Tsitsiklis. 2000. Actor-Critic Algorithms. In *SIAM Journal on Control and Optimization*. MIT Press, 1008–1014.

Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. 2014. Locomotion Control for Many-muscle Humanoids. *ACM Trans. Graph.* 33, 6, Article 218 (Nov. 2014), 11 pages. DOI: http://dx.doi.org/10.1145/2661229.2661233

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2015. End-to-End Training of Deep Visuomotor Policies. *CoRR* abs/1504.00702 (2015).

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015).

Libin Liu, Michiel van de Panne, and KangKang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Transactions on Graphics* 35, 3 (2016).

Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. 2012. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 154.

GE Loeb. 1995. Control implications of musculoskeletal mechanics. In *Engineering in Medicine and Biology Society, 1995., IEEE 17th Annual Conference*, Vol. 2. IEEE, 1393–1394.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust Physics-Based Locomotion Using Low-Dimensional Planning. *ACM Transactions on Graphics* 29, 3 (2010).

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel Todorov. 2015. Interactive Control of Diverse Complex Characters with Neural Networks. In *Advances in Neural Information Processing Systems 28*. 3132–3140.

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)* 35, 5 (2016). to appear.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR* abs/1506.02438 (2015).

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proc. International Conference on Machine Learning*. 387–395.

R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. 2001. Policy Gradient Methods for Reinforcement Learning with Function Approximation. (2001).

Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. 2014. Learning Bicycle Stunts. *ACM Trans. Graph.* 33, 4, Article 50 (2014), 50:1–50:12 pages.

Hado Van Hasselt. 2012. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*. Springer, 207–251.

Arthur J. van Soest and Maarten F. Bobbert. 1993. The contribution of muscle properties in the control of explosive movements. *Biological Cybernetics* 69, 3 (1993), 195–204. DOI: http://dx.doi.org/10.1007/BF00198959

Jack M. Wang, Samuel R. Hamner, Scott L. Delp, Vladlen Koltun, and More Specifically. 2012. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph* (2012).

Paweł WawrzyńSki and Ajay Kumar Tanwani. 2013. Autonomous reinforcement learning with experience replay. *Neural Networks* 41 (2013), 156–167.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.* 26, 3 (2007), Article 105.

SUPPLEMENTARY MATERIAL

Bounded Action Space

Properties such as torque and neural activation limits result in bounds on the range of values that can be assumed by actions for a particular parameterization. Improper enforcement of these bounds can lead to unstable learning as the gradient information outside the bounds may not be reliable [Hausknecht and Stone 2015]. To ensure that all actions respect their bounds, we adopt a method similar to the inverting gradients approach proposed by Hausknecht and Stone [2015]. Let $\nabla a = (a - \mu(s))\mathbb{A}(s,a)$ be the empirical action gradient from the policy gradient estimate of a Gaussian policy. Given the lower and upper bounds $[l^i, u^i]$ of the ith action parameter, the bounded gradient of the ith action parameter $\nabla \tilde{a}^i$ is determined according to

$$\nabla \tilde{a}^i = \begin{cases} l^i - \mu^i(s), & \mu^i(s) < l^i \text{ and } \nabla a^i < 0 \\ u^i - \mu^i(s), & \mu^i(s) > u^i \text{ and } \nabla a^i > 0 \\ \nabla a^i, & \text{otherwise} \end{cases}$$

Unlike the inverting gradients approach, which scales all gradients depending on proximity to the bounds, this method preserves the empirical gradients when bounds are respected, and alters the gradients only when bounds are violated.

Reward

The terms of the reward function are defined as follows:

$$r_{pose} = \exp\left(-||q^* - q||^2_W\right)$$

$$r_{vel} = \exp\left(-||\dot{q}^* - \dot{q}||^2_W\right)$$

$$r_{end} = \exp\left(-40 \sum_e ||x_e^* - x_e||^2\right)$$

$$r_{root} = \exp\left(-10(h^*_{root} - h_{root})^2\right)$$

$$r_{com} = \exp\left(-10||\dot{x}^*_{com} - \dot{x}_{com}||^2\right)$$

q and q^* denotes the character pose and reference pose represented in reduced-coordinates, while $\dot{q}$ and $\dot{q}^*$ are the respective joints velocities. W is a manually-specified per joint diagonal weighting matrix. h_{root} is the height of the root from the ground, and $\dot{x}_{com}$ is the center of mass velocity.

Sensitivity Analysis

We further analyze the sensitivity of the results to different initializations and design decisions. Figure 13 compares the learning curves from multiple policies trained using different random initializations of the networks. Four policies are trained for each actuation model. The results for a particular actuation model are similar across different runs, and the trends between the various actuation models also appear to be consistent. To evaluate the sensitivity to the amount of exploration noise applied during training, we trained policies where the standard deviation of the action distribution is twice and half of the default values. Figure 14 illustrates the learning curves for each policy. Overall, the performance of the policies do not appear to change significantly for the particular range of values. Finally, Figure 15 compares the results using different network architectures. The network variations include doubling the number of units in both hidden layers, halving the number of hidden units, and inserting an additional layer with 512 units between the two existing hidden layers. The choice of network structure does not appear to have a noticeable impact on the results, and the differences between the actuation models appear to be consistent across the different networks.

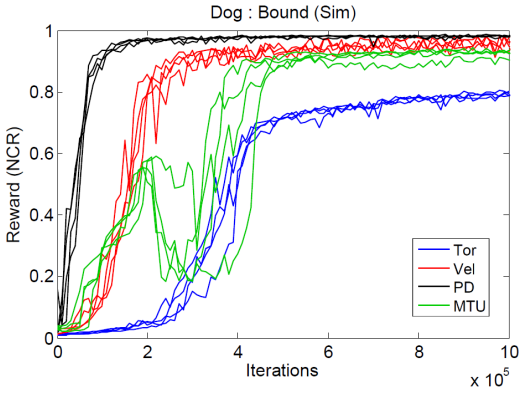

Figure 13: Learning curves from different random network initializations. Four policies are trained for each actuation model.

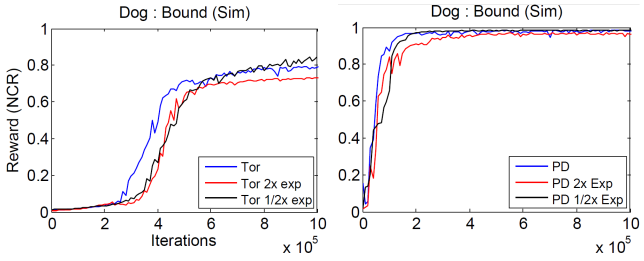

Figure 14: Learning curves comparing the effects of scaling the standard deviation of the action distribution by 1x, 2x, and 1/2x.

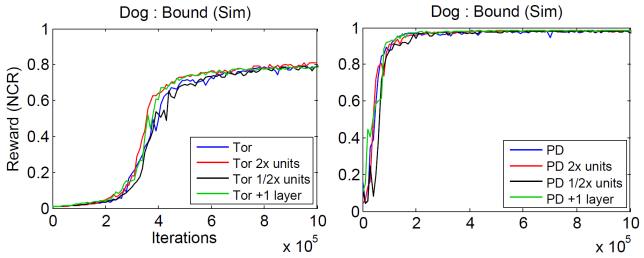

Figure 15: Learning curves for different network architectures. The network structures include, doubling the number of units in each hidden layer, halving the number of units, and inserting an additional hidden layer with 512 units between the two existing hidden layers.

While the MTU actuation parameters were optimized via automated actuator optimization, the actuation parameters for the other models, e.g. PD gains, were manually specified. To analyze the policies' sensitivity to the manually specified parameters, we trained PD policies with gains scaled by 0.25, 0.5, 1, and 2. Figure 16 shows the learning curves using the different sets of actuation parameters. The behaviour of the policies appears robust to changes within a factor of 2. Reducing the gains to 0.25 of their default values exhibits some negative impact to performance.

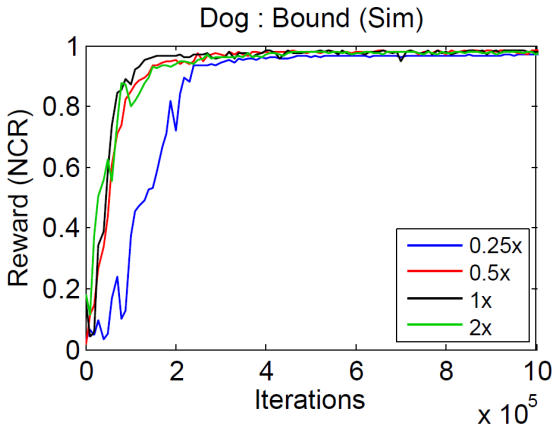

Figure 16: Learning curves for PD policies with different scalings of the PD gains.

To further evaluate the actuation models' performance with respect to different reference motions, we recorded a reference motion of the MTU policy for Dog : Bound (Sim), and retrained new policies to imitate the motion. Learnings curves are available in Figure 17. The performance of the various policies appear to respect the trends observed with other reference motions. Though the reference motion was recorded from an MTU policy, the PD policy still learns more quickly than the MTU policy.

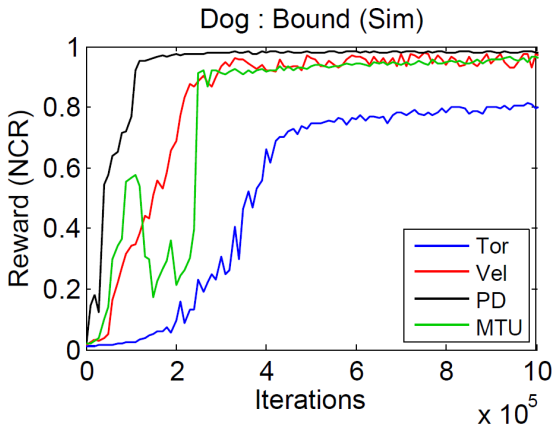

Figure 17: Learning curves for policies trained to imitate a reference motion recorded from an MTU policy.

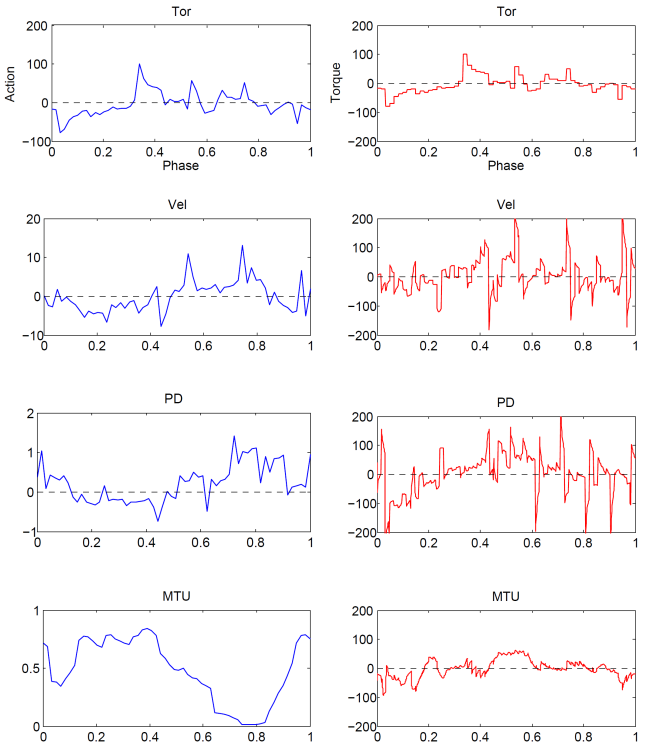

Figure 18: Policy actions over time and the resulting torques for the four action types. Data is from one biped walk cycle (1s). Left: Actions (60 Hz), for the right hip for PD, Vel, and Tor, and the right gluteal muscle for MTU. Right: Torques applied to the right hip joint, sampled at 600 Hz.

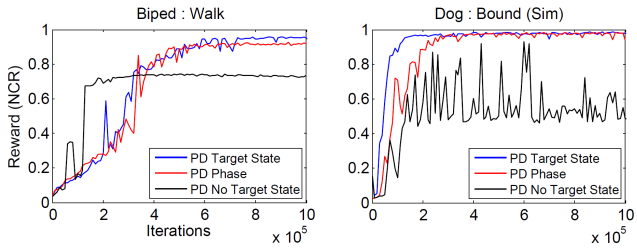

Figure 19: Learning curves for different state representations including state + target state, state + phase, and only state.

Parameter	Value	Description
γ	0.9	cumulative reward discount factor
α_π	0.001	actor learning rate
α_V	0.01	critic learning rate
momentum	0.9	stochastic gradient descent momentum
ϕ weight decay	0	L2 regularizer for critic parameters
θ weight decay	0.0005	L2 regularizer for actor parameters
minibatch size	32	tuples per stochastic gradient descent step
replay memory size	500000	number of the most recent tuples stored for future updates

Table 2: Training hyperparameters.

Character + Actuation Model	State Parameters	Action Parameters	Actuation Parameters
Biped + Tor	58	6	0
Biped + Vel	58	6	6
Biped + PD	58	6	12
Biped + MTU	74	16	114
Raptor + Tor	154	18	0
Raptor + Vel	154	18	18
Raptor + PD	154	18	36
Raptor + MTU	194	40	258
Dog + Tor	170	20	0
Dog + Vel	170	20	20
Dog + PD	170	20	40
Dog + MTU	214	44	282

Table 3: The number of state, action, and actuation model parameters for different characters and actuation models.

Character + Actuation	Motion	Performance (NCR)	Learning Speed (AUC)
Biped + Tor	Walk	0.7662 ± 0.3117	0.4788
Biped + Vel	Walk	0.9520 ± 0.0034	0.6308
Biped + PD	Walk	0.9524 ± 0.0034	0.6997
Biped + MTU	Walk	**0.9584 ± 0.0065**	**0.7165**
Biped + Tor	March	0.9353 ± 0.0072	0.7478
Biped + Vel	March	**0.9784 ± 0.0018**	0.9035
Biped + PD	March	0.9767 ± 0.0068	**0.9136**
Biped + MTU	March	0.9484 ± 0.0021	0.5587
Biped + Tor	Run	0.9032 ± 0.0102	0.6938
Biped + Vel	Run	**0.9070 ± 0.0106**	0.7301
Biped + PD	Run	0.9057 ± 0.0056	**0.7880**
Biped + MTU	Run	0.8988 ± 0.0094	0.5360
Raptor + Tor	Run (Sim)	0.7265 ± 0.0037	0.5061
Raptor + Vel	Run (Sim)	0.9612 ± 0.0055	0.8118
Raptor + PD	Run (Sim)	**0.9863 ± 0.0017**	**0.9282**
Raptor + MTU	Run (Sim)	0.9708 ± 0.0023	0.6330
Raptor + Tor	Run	0.6141 ± 0.0091	0.3814
Raptor + Vel	Run	0.8732 ± 0.0037	0.7008
Raptor + PD	Run	**0.9548 ± 0.0010**	**0.8372**
Raptor + MTU	Run	0.9533 ± 0.0015	0.7258
Dog + Tor	Bound (Sim)	0.7888 ± 0.0046	0.4895
Dog + Vel	Bound (Sim)	0.9788 ± 0.0044	0.7862
Dog + PD	Bound (Sim)	**0.9797 ± 0.0012**	**0.9280**
Dog + MTU	Bound (Sim)	0.9033 ± 0.0029	0.6825
Dog + Tor	Rear-Up	0.8151 ± 0.0113	0.5550
Dog + Vel	Rear-Up	0.7364 ± 0.2707	0.7454
Dog + PD	Rear-Up	**0.9565 ± 0.0058**	**0.8701**
Dog + MTU	Rear-Up	0.8744 ± 0.2566	0.7932

Table 4: Performance of policies trained for the various characters and actuation models. Performance is measured using the normalized cumulative reward (NCR) and learning speed is represented by the normalized area under each learning curve (AUC). The best performing parameterizations for each character and motion are in bold.