Learning a family of motor skills from a single motion clip

SEYOUNG LEE, Seoul National University, Korea
SUNMIN LEE, Seoul National University, Korea
YONGWOO LEE, Seoul National University, Korea
JEHEE LEE, Seoul National University, Korea

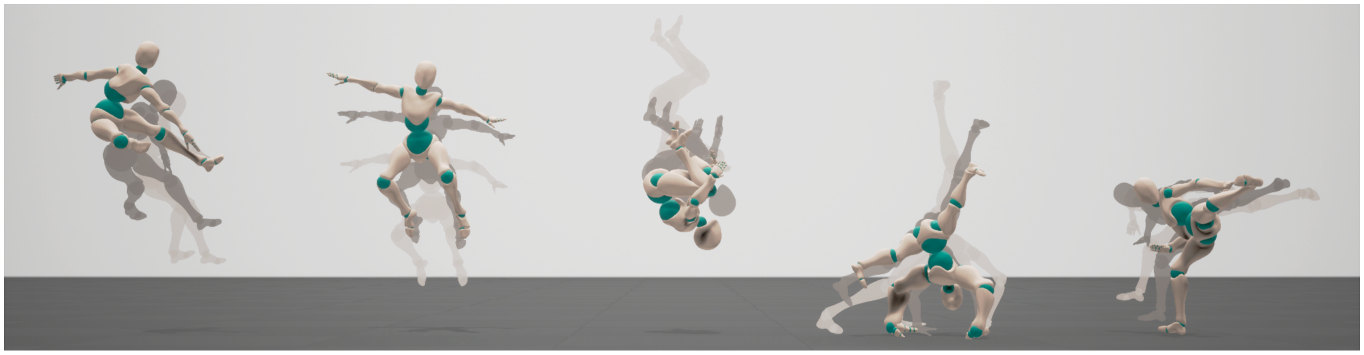
Fig. 1. The parameterized motor skills of obstacle jump, jump, backflip, cartwheel and kick motions.

We present a new algorithm that learns a parameterized family of motor skills from a single motion clip. The motor skills are represented by a deep policy network, which produces a stream of motions in physics simulation in response to user input and environment interaction by navigating continuous action space. Three novel technical components play an important role in the success of our algorithm. First, it explicitly constructs motion parameterization that maps action parameters to their corresponding motions. Simultaneous learning of motion parameterization and motor skills significantly improves the performance and visual quality of learned motor skills. Second, continuous-time reinforcement learning is adopted to explore temporal variations as well as spatial variations in motion parameterization. Lastly, we present a new automatic curriculum generation method that explores continuous action space more efficiently. We demonstrate the flexibility and versatility of our algorithm with highly dynamic motor skills that can be parameterized by task goals, body proportions, physical measurements, and environmental conditions.

CCS Concepts: • **Computing methodologies → Machine learning**; **Animation**.

Authors' addresses: Seyoung Lee, Department of Computer Science and Engineering, Seoul National University, Korea, seyounglee@mrl.snu.ac.kr; Sunmin Lee, Department of Computer Science and Engineering, Seoul National University, Korea, sunmin.lee@mrl.snu.ac.kr; Yongwoo Lee, Department of Computer Science and Engineering, Seoul National University, Korea, yongwoo.lee@mrl.snu.ac.kr; Jehee Lee, Department of Computer Science and Engineering, Seoul National University, Korea, jehee@mrl.snu.ac.kr.

Additional Key Words and Phrases: Learning Motor Skills, Physics-Based Simulation, Data-Driven Animation, Deep Reinforcement Learning

ACM Reference Format:
Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. 2021. Learning a family of motor skills from a single motion clip. *ACM Trans. Graph.* 40, 4, Article 93 (August 2021), 13 pages. https://doi.org/10.1145/3450626.3459774

1 INTRODUCTION

Humans can learn new motor skills from a demonstration. A student observes an instructor performing a punch and learns to imitate the movement by providing adequate actuation torques at joints. Furthermore, the student may also learn to repurpose the learned motor skill for striking in different directions and forces. The key technical challenge of such generalization is repurposing motor skills based on physical and biological principles. For example, to punch harder, it is advisable to pull the arm back more and then swing the arm faster to generate bigger momentum. Making a change to the task requires harmonious coordination of the arm swing trajectory and its timing, linear/angular momentum, and force/torque at joints. Previous approaches often exploited a comprehensive set of motions to describe how motor skills generalize. Collecting such large datasets requires considerable time and effort.

In this paper, we present a new approach to construct a parameterized family of motor skills from a single motion clip. Our algorithm generalizes a base motor skill, that mimics the reference motion, to generate a wide variety of skills that meet novel conditions and goals. The parameterized motor skills are represented by a deep policy network, which produces a stream of motions in physics simulation in response to user input and environment interaction by navigating continuous task space.

Specifically, we need to address two sub-problems. One is to learn a motor skill that mimics an input motion, and the other is to adapt the input motion in a physically valid manner for a range of conditions and tasks, which form a continuous task space. Previously, the former has been addressed using Deep Reinforcement Learning (DRL) [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2018], while the latter has been formulated as nonlinear trajectory optimization [Agrawal et al. 2014; Hong et al. 2019; Mordatch et al. 2012]. The brute-force algorithm would sample a condition/task parameter from the task space, run a trajectory optimization algorithm to adapt the input motion clip to meet the condition and run a DRL algorithm to learn a motor skill for the adapted motion. This process should repeat for a dense set of condition/goal parameters so the brute-force algorithm could be prohibitively slow for large task spaces. Our new algorithm achieves considerable performance improvements over the brute-force algorithm by effectively reusing experience tuples and planning task space exploration.

Our learning algorithm includes novel technical components. First, it explicitly constructs motion parameterization as a deep network in the learning process to allow a family of motor skills to be learned concurrently while sharing simulation experiences. The parameterization network also allows trajectory optimization to be incorporated seamlessly into reinforcement learning. Secondly, we adopt continuous-time reinforcement learning to explore temporal variations of the reference motion as well as its spatial variations. Thirdly, we present a new automatic curriculum generation method to explore continuous task space more efficiently. It rapidly explores unvisited regions in the task space, while steadily maintaining already visited regions not to forget previously learned motor skills.

We will demonstrate the flexibility and versatility of our algorithm with highly dynamic motor skills. Our algorithm provides great flexibility in choosing condition/task parameters, which we simply call *task parameters* for brevity, that can be selected from task goals (e.g., target position), body proportions (e.g., height and limb lengths), physical measurements and quantities (e.g., linear/angular velocities, linear/angular momentum, time duration, impact, kinetic energy, mass, inertia, gravity), and environment conditions (e.g., obstacle heights). Our characters are simulated and controlled interactively to perform a range of motor skills in the real-time physics simulation.

2 RELATED WORK

2.1 Data-driven Animation

High-quality motion capture data serve as building blocks of expressive character animation in movies, video games, and digital media. Each motion clip captures the motion of a specific subject in a specific environment and in a specific mood and style. There is a large array of literature that explores the adaptation and reuse of motion capture data to deal with new body proportions/structures [Aberman et al. 2020a; Gleicher 1998; Hecker et al. 2008; Lee and Shin 1999; Won and Lee 2019], new environments [Agrawal and van de Panne 2016; Lee et al. 2006], new styles [Aberman et al. 2020b; Liu et al. 2005; Yumer and Mitra 2016], and multi-character interactions [Ho et al. 2010; Kim et al. 2009; Won et al. 2014].

Large motion datasets are a valuable source of constructing continuous, parameterized spaces of motion. Blending a parameterized family of similar motions according to blending weights is a frequently used technique for character animation. RBF (Radial Basis Function) interpolation has been popular in early studies to deal with multivariate interpolation [Mukai and Kuriyama 2005; Park et al. 2002; Rose et al. 1998]. Learning-based techniques, such as GPLVM (Gaussian Process Latent Variable Model), allow more flexibility to learn a generative dynamic model from unorganized motion data [Grochow et al. 2004; Levine et al. 2012; Wang et al. 2007]. Recently, deep learning approaches prevail in constructing scalable generative models from large, unorganized motion datasets [Holden et al. 2020; Lee et al. 2018b; Park et al. 2019; Starke et al. 2020; Won et al. 2020].

2.2 Optimal Control and Trajectory Optimization

Trajectory optimization is the process of designing a complex articulated behavior that minimizes some performance measure, such as total joint torque and metabolic energy expenditure, while satisfying a set of constraints. Trajectory optimization can be reformulated to take an input motion such that it outputs a new constraint-satisfying trajectory that looks similar to the input motion. This reformulation leads to physics-based motion editing and retargeting [Abe et al. 2004; Majkowska and Faloutsos 2007; McCann et al. 2006; Sok et al. 2010]. Model predictive control (MPC), which solves trajectory optimization for a finite time-horizon repeatedly and execute only the first step of the trajectory to proceed, has also been studied in computer graphics literature [Hämäläinen et al. 2015; Hong et al. 2019]. Since the dynamics of articulated bodies are not smooth at collision and contact, derivative-free, sampling based optimization methods, such as CMA-ES (Covariant Matrix Adaptation Evolutionary Steps), have often been exploited to address trajectory optimization [Bharaj et al. 2015; Liu et al. 2015, 2012, 2010; Ye and Liu 2012]. The works of Agrawal *et al.* [2014] also use CMA-ES to generate a diverse set of motions from a single motion which is manually broken into multiple control phases.

2.3 Policy Learning

Designing the control system of an articulated body model has been a long-standing challenge in computer graphics. Recent DRL-based policy learning approaches have achieved significant improvements in terms of stability, performance and scalability of control. DRL-based continuous control is not encumbered by the structure of the dynamics system and the type of motor skills, so it has successfully been exploited to deal with biped locomotion [Heess et al. 2017; Peng et al. 2017], highly-dynamic sports action [Yu et al. 2019], object manipulation [Clegg et al. 2018], quadruped locomotion [Luo et al. 2020; Peng et al. 2019], anatomical bodies [Jiang et al. 2019; Lee et al. 2019], aerobatic flapping flight [Won et al. 2017, 2018], underwater swimming [Min et al. 2019], and synthetic vision guidance [Merel et al. 2020].

Although DRL is capable of learning complex motor skills from scratch without any reference motion to imitate [Heess et al. 2017], the best human-like motor skills are often achieved when high-quality motion capture data are provided as a reference [Peng et al. 2018]. Recent studies demonstrated that DRL could learn an integrated control policy equipped with many heterogeneous motor

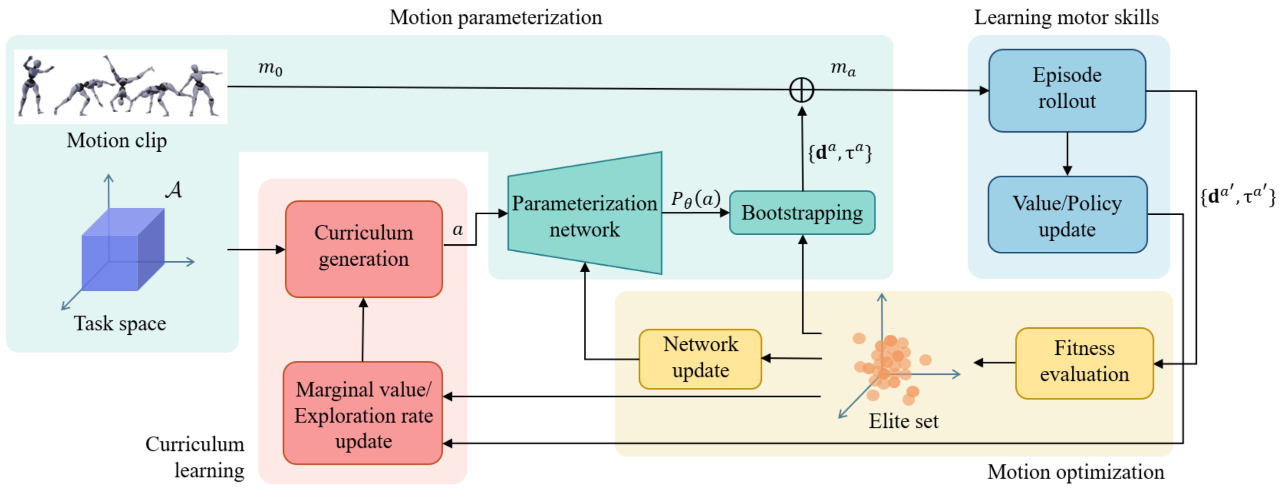

Fig. 2. System overview

skills from data-driven motion generators [Bergamin et al. 2019; Park et al. 2019]. Won *et al.* [2020] demonstrated the scalability of DRL by learning a large-scale control policy from eight hours of motion data.

The parameterization of motor skills allows the control system to be more versatile and accurate. Reinforcement learning is inherently capable of providing some form of generalization through state space exploration and function approximators. Previous studies in computer graphics demonstrated a locomotion/jump policy with a range of steering angles/jump heights learned from a single motion clip, though the scope of successful generalization is usually limited to a modest range [Lee et al. 2019; Luo et al. 2020; Peng et al. 2018]. Curriculum learning that presents the training data to the learning algorithm in order of increasing complexity is often useful for exploring the continuous parameter space [Bengio et al. 2009; Yu et al. 2018]. The idea of automatically generating a curriculum according to the progress of learning was also studied and applied to the control of physics-based characters [Matiisen et al. 2019; Won and Lee 2019]. Xie *et al.* [2020] evaluated several curriculum choices for the control of stepping-stone locomotion. Previous studies showed that incorporating trajectory optimization methods into policy learning frameworks can facilitate faster exploration and learning [Levine and Koltun 2013; Mordatch and Todorov 2014; Won et al. 2018].

3 MOTION PARAMETERIZATION

The reference motion $\mathbf{m}(t)$ is represented by a sequence of full-body poses at discrete time instances. Given reference time $\phi \in [0, T]$, the motion $\mathbf{m}(\phi)$ can be considered as a continuous, piecewise linear function that maps time to full-body pose. The poses at discrete time instances interpolate linearly to construct inbetween poses in the continuous time domain. The full-body pose of an articulated figure is represented by a heterogeneous array $(p_0, q_1, \cdots, q_L)$, where $p_0 \in R^3$ and $q_1 \in S^3$ are the position and orientation of the body root, respectively, and the other unit quaternions q_i's for $i > 1$ are joint angles.

The motion is supposed to take place under certain body conditions (e.g., height, weight, and maximum strength at joints) and physical conditions (e.g., gravity and friction coefficients), and may have task goals (e.g., jump height and punch targets). We can construct task space $\mathcal{A}$ by selecting several parameters of interest from body conditions, physical conditions, and task goals. The reference motion corresponds to the origin of the task space, and each parameter axis corresponds to the change in motion according to the parameter change. Each parameter vector $a \in \mathcal{A}$ corresponds to a parametrically-varied motion m_a, which looks similar to the reference motion but satisfies new conditions and tasks represented by a. This mapping constructs a parameterized family of motions $\mathcal{M} = \{\mathbf{m}_a | a \in \mathcal{A}\}$. Hereafter, we denote the reference motion by $\mathbf{m}_0$.

The spatiotemporal variant $\mathbf{m}_a$ of the reference motion is represented by motion displacement mapping and time warping.

$$\mathbf{m}_a(t) = (\mathbf{m_0} \oplus \mathbf{d})(\phi(t)), \tag{1}$$

where $\mathbf{d}(\phi) = (d_0(\phi), \cdots, d_L(\phi))$ adds motion displacements to the reference motion to modify its spatial trajectory and joint angles such that

$$\mathbf{m_0} \oplus \mathbf{d} = (p_0 + d_0, q_1 \exp(d_1), \cdots, q_L \exp(d_L)), \tag{2}$$

where $d_0 \in R^3$ is the linear displacement of the root position, d_1 is its angular displacement, and d_i for $i > 1$ is the angular displacement of a joint. Please refer to [Lee and Shin 1999] for a detailed description of motion displacement mapping. The time warp function $\phi(t)$ reparameterizes the motion to accelerate, deaccelerate, and change the timing of the task (see Figure 3). The time warp should be strictly monotonic everywhere to avoid traveling backward and stalling in time. We construct a monotonically increasing function by accumulating positive time increments $\exp(\tau_k)$ such that

$$\phi_i = \sum_{0 \le k \le i-1} \exp(\tau_k), \tag{3}$$

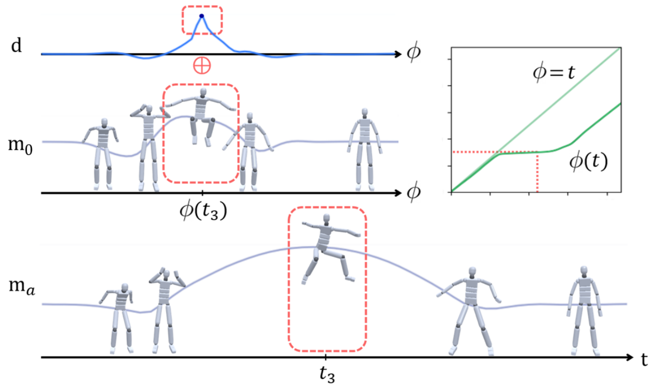

Fig. 3. Visualization of motion displacement mapping and time warping. The gray curves represent the y position of the pelvis over time in the Jump motion, and the blue curve represents the motion displacement between the two gray curves. The graph in the upper right corner shows the mapping between ϕ and t.

where $\phi_0 = 0$. $\exp(\tau_k)$ is always positive regardless of the value of τ_k. From the discrete samples ϕ_i, a continuous, piecewise linear, monotonic function $\phi(t)$ can be constructed by linear interpolation. Each motion $\mathbf{m}_a$ is represented by the trace of motion displacements and time increments over the duration of the motion.

Motion parameterization is constructed explicitly by learning a multilayer perceptron that maps parameter a to motion $\mathbf{m}_a$. We will discuss the learning procedure in the following sections.

4 LEARNING MOTOR SKILLS

It has been demonstrated in previous studies that deep reinforcement learning can learn a motor skill that mimics input motion capture data in physics simulation [Peng et al. 2018]. The motor skill is represented by a deep policy network π that generates actions (e.g., joint torques, muscle activations, or PD targets) at any states (e.g., joint angles, end-effector positions, and/or proprioceptive sensing). Our learning algorithm is also based on imitation learning but fundamentally different from the previous formulation in three aspects. First, our algorithm learns a policy network π_a that represents (infinitely many) motor skills parameterized by $a \in \mathcal{A}$. Second, parameterized motions $\mathbf{m}_a$ to imitate are not known at the beginning of learning. Our algorithm learns both parameterized motions $\mathbf{m}_a$ and their motor skills π_a simultaneously. Lastly, we adopt continuous-time reinforcement learning to allow the timing of parameterized motions to deviate from the reference motion by learning their time warp functions explicitly. In this section, we will discuss how these issues are dealt with.

4.1 State and Action

The *state* of the agent includes the positions and velocities of joints/end-effectors/task goals with respect to a local, moving coordinate system, and the height and up-vector of the pelvis (the root of the articulated tree) with respect to a world, reference coordinate system. The state also includes task parameter a, the current reference time ϕ, and a full-body pose (e.i. joint positions and velocities) at the next frame $\phi + \Delta\phi$. As discussed by Park et al. [2019], the rich state description including its future prediction supplements a scalar value ϕ to disambiguate phases in long motion data. In our formulation, the future frame from the reference motion serves as a future prediction.

The *action* of the agent is defined by spatial displacement $\mathbf{d}$ and time increment τ.

$$(\mathbf{d}_t, \tau_t) = \pi_a(s_t). \tag{4}$$

This action makes transition to its subsequent state by advancing reference and simulation time $\phi' = \phi + \exp(\tau_t)$ and $t' = t + \Delta t$, where Δt is constant. The state update by spatial displacement uses PD control in physics simulation, where the displaced pose $\mathbf{m}_0(\phi') \oplus \mathbf{d}_t$ serves as a PD target to generate joint torques. Advancing the dynamics simulation by time Δt updates the state at the next time instance.

4.2 Reward

The goal of reinforcement learning is to find the optimal policy that maximizes the expected cumulative reward. If the agent follows the optimal policy π_a^* in physics simulation, it will act out the reference motion approximately while satisfying the conditions and tasks denoted by a. The trace of spatial displacements and time increments forms the optimal motion $\mathbf{m}_a^*$ for parameter a.

The *reward* includes two terms for motion tracking and conditions/tasks enforcement.

$$r = w_{\text{tracking}} r_{\text{tracking}} + w_{\text{task}} r_{\text{task}}, \tag{5}$$

The agent receives tracking rewards r_{tracking} if it imitates the reference motion $\mathbf{m}_a$ accurately, while it receives task rewards r_{task} if it achieves the conditions/tasks represented by parameter a.

The tracking reward includes four sub-terms

$$r_{\text{tracking}} = w_q r_q + w_{\text{ee}} r_{\text{ee}} + w_{\text{com}} r_{\text{com}} + w_{\text{time}} r_{\text{time}}, \tag{6}$$

which respectively penalize the discrepancies in joint angles, end-effector positions, COM (center of mass), and time between the reference motion and simulation.

$$\begin{aligned}
r_q &= \exp\Big(-\frac{1}{\sigma_q^2} \sum_{i \in \text{joints}} \| \log(q_i^{-1} \hat{q}_i) \|^2 \Big) \\
r_{\text{ee}} &= \exp\Big(-\frac{1}{\sigma_{\text{ee}}^2} \sum_{j \in \text{ee}} \| \hat{x}_j - x_j \|^2 \Big) \\
r_{\text{com}} &= \exp\Big(-\frac{1}{\sigma_{\text{com}}^2} \| \hat{x}_{\text{com}} - x_{\text{com}} \|^2 \Big) \\
r_{\text{time}} &= \exp\Big(-\frac{1}{\sigma_{\text{time}}^2} \| (\exp(\hat{\tau}_t) - \exp(\tau_t) \|^2 \Big),
\end{aligned} \tag{7}$$

where $q_j \in S^3$ are joint angles, $x_j \in R^3$ are joint positions, $x_{\text{com}} \in R^3$ is the center of mass of the full-body. The hat symbol indicates measurements from $\mathbf{m}_a$. In our experiments, weights are common for all examples: $\sigma_q = 3.16, \sigma_{\text{ee}} = 0.8, \sigma_{\text{com}} = 0.35, \sigma_{\text{time}} = 0.1, w_q = 0.28, w_{\text{ee}} = 0.28, w_{\text{com}} = 0.28$, and $w_{\text{time}} = 0.16$.

The task reward is defined by a similar form.

$$r_{\text{task}} = \exp\Big(-\frac{1}{\sigma_{\text{task}}^2} \| \hat{x}_{\text{task}} - x_{\text{task}} \|^2 \Big), \tag{8}$$

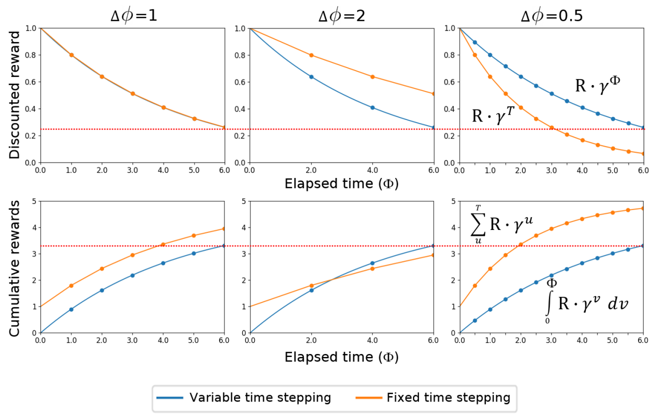

Fig. 4. The elapsed time and the reward graphs. T is elapsed simulation time, and Φ is elapsed reference time. The reward is received once in the first row and the reward is added every time step in the second row. The red lines are the baseline discounted reward and cumulative rewards at $\Phi = 6$ when time step $\Delta\phi = 1$, respectively. With our variable time step policy update, the discounted reward and cumulative rewards are invariant under reference time step. In this figure, we set $R = 1$ and $\gamma = 0.8$.

where x_{task} can be any measurement taken from the simulation including position, orientation, linear/angular velocity, linear/angular momentum, time duration, impact, pressure, and kinetic energy for a certain duration. We can have great flexibility in choosing parameters and defining their corresponding rewards. Any parameter with a zero range serves as a constraint. We use constraints to specify invariant factors that a family of parameterized motions should commonly satisfy.

There are two types of rewards: *continuous* and *spike*. The *continuous* rewards are received uninterruptedly over a certain period of time, while the *spike* reward is received instantly when a certain condition is satisfied. All tracking rewards are continuous. The task reward can be either of the two types. A reward defined over a duration (e.g., total joint torque or total kinetic energy during a time interval) is *spike* because the reward is received instantly when the simulation reaches the end of the duration. In our experiments, $w_{\text{tracking}} = 1.0$ and $w_{\text{task}} = 10.0$ for spike rewards, and $w_{\text{tracking}} = 0.9$ and $w_{\text{task}} = 0.1$ for continuous rewards. $w_{\text{tracking}} = 1.0$, if the task is not defined.

Physical quantities (e.g., mass, inertia, gravity) and body proportions (e.g., limb length) can also be chosen to form motion parameterization. In this case, it is not necessary to define corresponding rewards, but the changes in quantities and proportions are reflected in physics simulation and state transitions.

4.3 Variable Time-Step Policy Update

The learning algorithm is episodic. It generates many episodes of physics simulation tracking parameterized motions and collects experience tuples to update the value and policy networks. The Proximal Policy Optimization (PPO) is a popular policy-gradient method that addresses discrete-time Markov decision problems with uniform time steps. We derive its variant to address variable time-step problems. The length of time steps critically affects how rewards are discounted over time and how generalized advantages are estimated. Given a time step $\Delta\phi = \exp(\tau_t)$, the discount factor during $[\phi, \phi + \Delta\phi]$ is $\gamma^{\Delta\phi}$. Similarly, generalized advantage estimation (GAE) parameter during the time interval is $\lambda^{\Delta\phi}$. Provided that both continuous rewards r_c and spike rewards r_s are given, GAE with variable time stepping is

$$\delta(\phi) = \int_0^{\Delta\phi} \gamma^v R_c(\phi + v)dv + R_s(\phi) + \gamma^{\Delta\phi} V(\phi + \Delta\phi) - V(\phi)$$
$$A(\phi) = \delta(\phi) + \gamma^{\Delta\phi}\lambda^{\Delta\phi} A(\phi + \Delta\phi). \tag{9}$$

Here, the continuous rewards are integrated and discounted over the time interval, while the spike rewards are estimated instantly. This modified PPO algorithm makes policy learning invariant under time warping and therefore prevents $\mathbf{m}_a$ from unintentionally speeding up or slowing down regardless of their conditions/tasks (see Figure 4).

5 MOTION OPTIMIZATION

Finding any motion $\mathbf{m}_a$ that satisfies the conditions/tasks given by a while minimizing the deviation from the reference motion $\mathbf{m}_0$ can be formulated as trajectory optimization. The brute-force construction of motion parameterization requires solving this trajectory optimization for a continuous domain $a \in \mathcal{A}$. In this work, we do not optimize individual motions one-by-one, but learn the parameter-to-motion mapping gradually in a supervised manner. Since we reuse simulation episodes generated during reinforcement learning, motion parameterization can be constructed at a fraction of the computation cost of reinforcement learning.

5.1 Fitness

Each simulation episode generated by policy π_a forms the trace of motion displacements and time increments $\{(\mathbf{d}_t, \tau_t) | t = 0, \cdots, T\}$, which can be converted into a motion using Equation (1). We align the root trajectory of the episode with that of the $\mathbf{m}_0$ before converting. Though the goal a is pursued in the simulation, many episodes do not achieve this intended goal if the policy is sub-optimal. Since we are dealing with a continuous target domain, we use the insight of hindsight experience replay to reuse simulation rollouts [Andrychowicz et al. 2017]. The sub-optimal episode is useful if we retrospectively imagine that the agent was trying to achieve the goal $a' \in \mathcal{A}$ they actually ended up with. This episode can be used to learn the correlation between a' and $\mathbf{m}_{a'}$ if the episode represents a quality motion. The motion quality is measured using a fitness function

$$F = f_{\text{pose}} \cdot f_{\text{velocity}} \cdot f_{\text{contact}} \cdot f_{\text{drag}} + w_{\text{constraint}} \cdot r_{\text{constraint}}, \tag{10}$$

which compares the poses, velocities, and contact states of the episode to the reference motion $\mathbf{m}_0$.

$$f_{\text{pose}} = \exp\Big(- \frac{1}{\sigma^2_{\text{pose}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{i \in \text{joints}} \parallel \log(q_i(\phi)^{-1} \bar{q}_i(\phi)) \parallel^2 \Big)$$

$$f_{\text{velocity}} = \exp\Big(- \frac{1}{\sigma^2_{\text{velocity}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{i \in \text{joint}} \frac{\parallel \bar{v}_i(\phi) - v_i(\phi) \parallel^2}{\max(V, \parallel \bar{v}_i(\phi) \parallel^2)} \Big)$$

$$f_{\text{contact}} = \exp\Big(- \frac{1}{\sigma^2_{\text{contact}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{j \in ee} \bar{C}_j(\phi) \parallel \bar{y}_j(\phi) - y_j(\phi) \parallel^2 \Big)$$

$$f_{\text{drag}} = \exp\Big(- \frac{1}{\sigma^2_{\text{drag}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{j \in \text{ee}} \bar{C}_j(\phi) \bar{C}_j(\phi - 1) \parallel T_j(\phi) \parallel^2 \Big),$$

where q and v are joint positions and velocities with respect to the body local coordinate system. y is the height of the end-effector from the ground surface. $C_j(\phi) \in \{0, 1\}$ is a Boolean flag that indicates the contact state of the j-th joint at time ϕ. $T_j(\phi)$ is the translation of the end-effector in the xz plane. The bar over a symbol indicates that the value is measured from the reference motion $\mathbf{m}_0$. V is a constant that prevents vanishing denominators. The fitness function takes into account constraints (zero-range parameters) to evaluate if the episode satisfies the desired qualities specified by the user. In our experiments, the weights are $\sigma_{\text{pose}} = 2.67$, $\sigma_{\text{velocity}} = 75.5$, $\sigma_{\text{contact}} = 1.3$, $\sigma_{\text{drag}} = 0.7$, $w_{\text{constraint}} = 0.1$ and $V = 0.5$.

5.2 Parameterization Network

The policy learning algorithm in the previous section generates a large number of good episodes with its fitness value above a user-specified threshold. These episodes serve as training data for learning a parameterization MLP P_θ that takes the parameter a as input and outputs $\mathbf{m}_a$ such that

$$P_\theta(a) = \{(\mathbf{d}_t, \tau_t) | t = 0, \cdots, T\}. \tag{11}$$

The MLP is a simple regression network that learns the nonlinear mapping between parameters and motions minimizing the loss

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_t \|\mathbf{d}_t - \mathbf{d}^*_t\|^2 + (\tau_t - \tau^*_t)^2, \tag{12}$$

which measures the discrepancy between training data and network outputs. The MLP is learned in parallel with policy learning.

Since the performance of the MLP largely depends on the quality of its training data, feeding the MLP learning with consistently high fitness episodes is necessary. To do so, we maintain a list of elite episodes E. Since policy learning generates simulation episodes in an arbitrary order, it is important to ensure that new episodes do not override previous ones with higher fitness values. Whenever a new, above-threshold episode with parameter a and fitness F is generated, we compare it with its neighbors within radius r in the elite set. Let a' and F' be the parameter and fitness of a neighbor, respectively. The weighted distance between a and a' is within r

$$(a - a')^\top A (a - a') < r^2 \tag{13}$$

in the neighborhood, where A is a diagonal weight matrix. The radius r controls the granularity of samples in the elite set. In our experiments, we set $A = I$ and $r = 0.1$. If the new episode is better than any of its neighbors ($F > F'$), the new episode will be used to update the parameterization network and be included in the elite set to replace the inferior neighbors. Otherwise, the new episode is dismissed. In this way, every episode in the elite set is the best in its neighborhood, and the parameterization network is progressively updated with a series of monotonically-improving training data. We experimentally found that the quality of the converged motion is not sensitive to the threshold of fitness. In our experiments, we set the value 0.1 to exclude completely failed episodes. The elite set is stored in a k-d tree so that neighborhood searches can be performed efficiently.

5.3 Bootstrapping

In the early stages of learning, the premature parameterization network may generate low-quality motions, which negatively affect subsequent motor skill learning. To bootstrap network learning, two strategies are adopted. First, we start by training a tracking controller (a.k.a. policy π_0) that mimics the original reference m_0 until the learning curve plateaus. This base policy provides a good starting point to explore similar motor skills in motion parameterization. Second, we use a more reliable proxy to replace the network outputs $P_\theta(a)$ until the regression loss decreases below the threshold. The proxy is computed by locally linear regression of elite episodes. To do so, we take k-nearest neighbors of a in the elite set E and linearly interpolate them with weights

$$w_k = F_k \cdot \exp\big(- \frac{1}{\sigma_k^2} (a_k - a)^T A (a_k - a) \big). \tag{14}$$

The weights take into account both distance in the task space and fitness values. The k-nearest neighbor interpolation always generates decent quality motions because the neighbors to be interpolated are chosen from the elite set. We set $k = 5$, $\sigma_k = 0.28$ in our experiments.

Once the regression loss goes below the threshold, the system chooses outputs randomly between direct network outputs and locally-linear estimates. In this way, motor skill learning is consistently provided with high-quality motions to imitate while progressively learning motion parameterization.

6 CURRICULUM GENERATION

The order in which the learning algorithm explores the task space is important because motor skills are learned over a continuous domain. Uniform random sampling often fails when the RL algorithm tries to learn motor skills far away from the reference motion. A popular approach to overcome this problem is to use curriculum learning [Bengio et al. 2009]. The policy for harder (far from the reference motion) motor skills is discoverable after mastering easier (near the reference motion) motor skills. Curriculum learning suggests that the learning algorithm needs to explore the task space in a near-to-far order, mastering easy ones and proceeding to harder ones. This simple curriculum needs elaboration to address two issues. First, mastering a (even easy) motor skill is computationally demanding. Mastering one by one to cover the entire task space is impractical, if not impossible. Second, even if motor skills are learned up to a certain threshold in the early stages of learning, policies may forget the skills while learning the others. Perhaps, mastering individual motor skills early on is not necessary. More

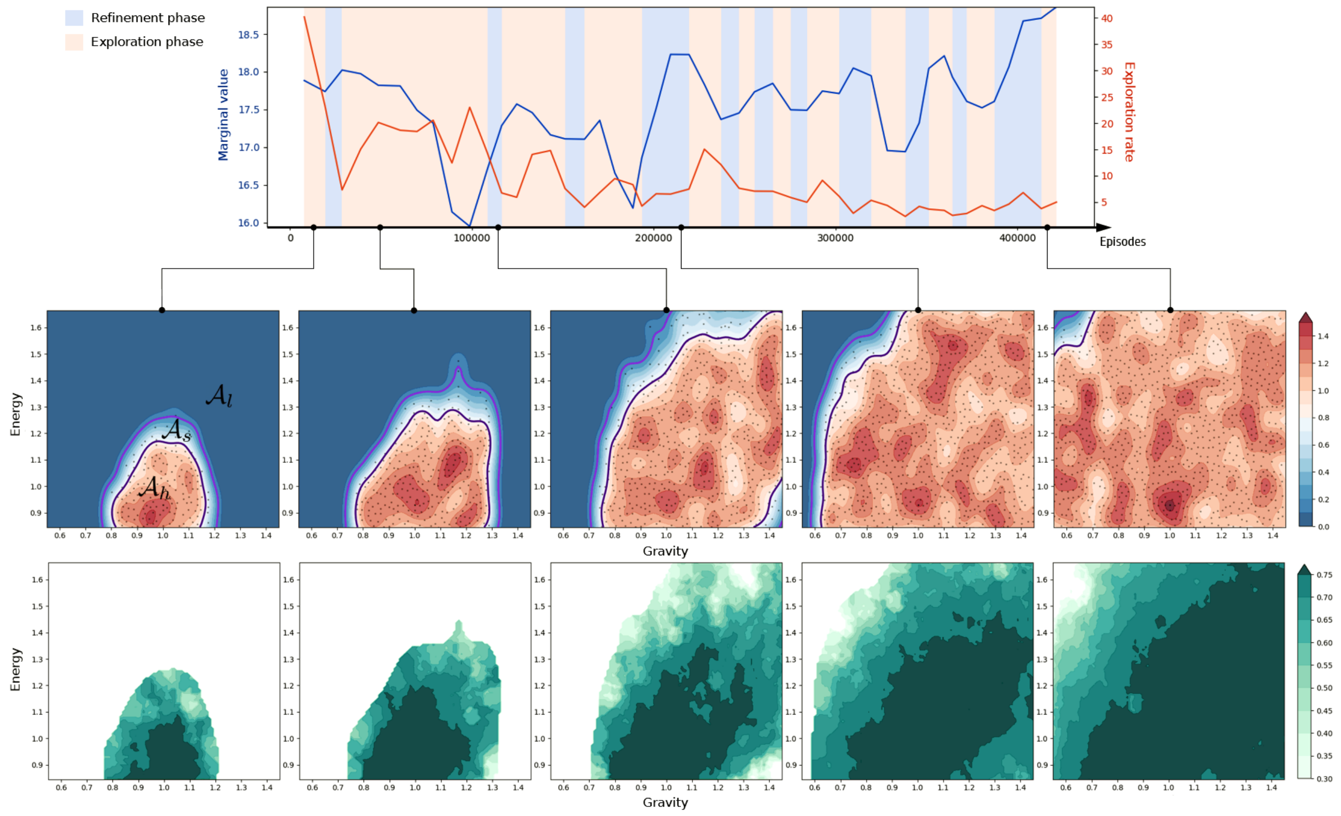

Fig. 5. Curriculum of learning jumps parameterized by gravity and kinematic energy. (Top) The blue and red plots, respectively, represent the change in mean marginal value and exploration rate as learning progresses. The peach and blue shades, respectively, represent the exploration and refinement phases. (Middle) The elite episodes are shown as black dots and their density is shown as red-to-blue shades. (Bottom) The fitness values of the elite episodes.

importantly, easier ones should be mixed in while learning harder ones to avoid forgetting.

In this section, we present a new automatic curriculum generation method that allows the learning algorithm to rapidly explore unvisited regions in the task space while continuously refining already visited regions not to forget (see Figure 5). The method alternates between exploration phases and refinement phases. A batch of action parameters are picked from less visited regions in exploration phases, whereas they are picked from heavily visited regions in refinement phases. In our experiments, each batch includes 20 task parameters. The motor skill learner produces 30 simulation episodes for each parameter and updates the value/policy networks based on the episodes. We estimate the density at parameter $a \in \mathcal{A}$ by kernel density estimation.

$$\text{density}(a) \propto \sum_{a_k \in \text{neighbor}(a)} \exp\left(-(a_k - a)^T A (a_k - a)\right), \tag{15}$$

If the density is above a user-specified threshold, parameter a is in the heavily visited region $\mathcal{A}_h \subset \mathcal{A}$. Otherwise, it is in the less visited region $\mathcal{A}_l = \mathcal{A} \setminus \mathcal{A}_h$.

6.1 Marginal Value and Exploration Rate

Catching up with how well each task parameter is achieved and deciding when to make transitions between the phases are the keys to robust and efficient learning. Ideally, the method should be independent of the choice of reference motions and task parameters and should not require careful parameter tuning. We use two measures, *marginal value* and *exploration rate*. The state of the agent is high-dimensional, and only a few of the dimensions are selected to form task space. Let state $s = (s_a, s_b)$ be a composition of task parameters s_a and the rest of the parameters s_b. Marginalizing s_b, the value function is reduced to

$$\bar{V}(s_a) = \int_{s_b} V(s_a, s_b)\, ds_b. \tag{16}$$

Here, we assume uniform density of s_b. The marginal value function $\bar{V}(s_a)$ estimates how much rewards will be received if the agent follows the current policy pursuing task a. The *mean marginal value* over $\mathcal{A}$ is $\bar{V}(\mathcal{A}) = E_{a \in \mathcal{A}}(\bar{V}(s_a))$.

The *exploration rate* estimates how successful the rollout of simulation episodes was in discovering new motions and improving motor skills. We define the exploration rate by a number of new episodes added to the elite set for a single batch. To prevent the exploration rate from increasing meaninglessly, the exploration rate increases when the fitness of new episode is clearly higher than those of dropped ones. We set the threshold 0.02.

6.2 Exploration and Refinement

The goal of exploration phases is the expansion of $\mathcal{A}_h$ by visiting its surrounding areas. To explore the task space gradually in a near-to-far order, a batch of task parameters B are sampled in the surrounding area $\mathcal{A}_s$ where the density is between D' and D for $D' < D$. $\mathcal{A}_s$ is part of $\mathcal{A}_l$ and adjacent to $\mathcal{A}_h$. In exploration phases, the exploration rate is proportional to the mean marginal value $\bar{V}(B)$, because the control policy is likely to improve if many good episodes are discovered. Similarly, the exploration rate and the average fitness value tend to be proportional to each other. D and D' decide how aggressively or conservatively we want the explore. In our experiments, $D = 0.8$ and $D' = 0.2$.

The goal of refinement phases is to improve the policy gradually and evenly across $\mathcal{A}_h$ while avoiding forgetting. We use an MCMC (Markov Chain Monte Carlo) method with multiple starting points [Won and Lee 2019] to sample a batch of task parameters in $\mathcal{A}_h$. The MCMC algorithm with a probability distribution

$$p(s_a) = \exp(-k\frac{\bar{V}(s_a) - \mu}{\mu}) \tag{17}$$

samples more where the marginal value is low and therefore continuously pursues a state where marginal values are evenly distributed across $\mathcal{A}_h$. In our experiments, the constant k is 10.

6.3 Transition Criteria

Our learning algorithm monitors the change of exploration rates in $\mathcal{A}_h$ and its surrounding area $\mathcal{A}_s$ by occasionally taking samples. The exploration rate tends to decrease in exploration phases. The transitioning to a refinement phase occurs when the exploration rate in $\mathcal{A}_s$ becomes smaller than the exploration rate in $\mathcal{A}_h$. In refinement phases, the samples drawn from $\mathcal{A}_h$ influence not only $\mathcal{A}_h$ but also its surrounding areas $\mathcal{A}_s$. Therefore, the mean marginal value and the exploration rate in $\mathcal{A}_s$ improves gradually during refinement. The transitioning to an exploration phase occurs when the exploration rate in $\mathcal{A}_s$ becomes larger than the exploration rate in $\mathcal{A}_h$.

7 EXPERIMENTS

Our simulation system is written in C++ and based on DART dynamics toolkit [Lee et al. 2018a]. Reinforcement learning and regression networks are written in Python and based on Tensorflow library [Abadi et al. 2016]. The base model of our animated characters is 1.65 m tall and weighs 61.6 kg consisting of rigid bones connected by 21 ball-and-socket joints. The articulated skeleton is actuated by joint torques, which are computed by linear-time stable PD controllers [Yin and Yin 2020]. The PD gains are $k_p = 600$ and $k_d = 49$. The simulation time step is 150 Hz, and the control time step is 30 Hz.

The policy network has four fully-connected layers of 1024 ReLU nodes. The value network and the parameterization network have two fully-connected layers of 512 ReLU nodes. The clipping range of PPO is 0.2, the learning rate of the policy function is 0.0002, the learning rate of value/parameterization networks is 0.001, the discount factor is $\gamma = 0.95$, and the GAE parameter is $\lambda = 0.95$. The minibatch size of the policy and value networks is 1024. The minibatch size of the marginal value and parameterization network is 128. The parameterization network is updated 10 times every ten policy updates. The MCMC algorithm draws 1000 samples for every five policy updates.

We used motion data available on public motion databases including Mixamo [Adobe Systems Incs. 2018] and CMU motion databases [CMU 2002]. The computation time for learning varies depending on the dimension and range of task spaces. It takes 12 hours to 72 hours on a single PC with AMD Ryzen 9 3950x (3.5 GHz, 16 cores). Since sequential computation for physic simulation is hard to benefit from the parallelization power of GPU, most of the computation is done in the CPU.

7.1 Motor Skills and Their Parameterization

We conducted experiments with various motor skills. The task parameters, hyperparameters, and performance statistics are summarized in Table 1. The total number N of episodes the algorithm generated measures computational efficiency. ER (exploration ratio), which is the percentage of the heavily visited regions in the task space, measures how successful the algorithms are in exploring the task space. The unit of each task parameter is a multiple of that in the reference data. Exceptionally, the unit of angle parameter is radian, and the unit of height parameter in BoxJump is meter. Some reward terms and fitness terms look similar and are sometimes interchangeable. The rule of thumb is that the reward function is designed to learn motor skills. The reward function includes essential terms that are necessary for all examples. In contrast, the fitness function is designed to improve the visual quality of simulation results and can be customized for individual examples.

The Jump example takes two task parameters: gravity and kinetic energy (see Figure 5). The X-axis of the task space represents the gravity ranging from $0.5G$ to $1.65G$. The Y-axis represents the kinetic energy ranging from $0.9E$ to $1.7E$. Here, G is the gravity on Earth, and E is the full-body kinetic energy at the onset of jump in the reference motion. Higher kinetic energy and lower gravity result in higher jumps.

The SpinKick example has two task parameters: kick height and angular momentum about the vertical axis. The kick height is 1.5 meters, and the spinning angle is 6.0 radians in the reference motion. The task space spans the range of $[0.75, 2.25]$ meters and $[3.0, 12.0]$ radians. The character spins in the air in the reference motion.

The Punch skills parameterized by four parameters (distance, height, angle, impact) allows the character to throw a punch in the desired position with the desired power. We measure the impact between the fist and the target object by actually placing the object at the strike point predicted from the current trajectory in the physics simulation. The after-impact momentum of the object estimates the punch impact.

The Backflip skills are parameterized by jump height and angular velocity. The angular velocity parameter is calculated as the average angular velocity of the torso during the body is in the air. Since

Table 1. Hyperparameters and performance statistics.

skills	task parameters	range	constraints	N	ER
Backflip	angular velocity	[0.5, 2.0]	angular momentum rotation	2.1M	0.9
	height	[0.8, 1.3]			
BoxJump	distance	[1.0, 2.15]	-	2.2M	0.91
	height	[-0.9, 0.9]			
Cartwheel	length of arms	[0.65, 1.65]	rotation	3.5M	1
	length of legs	[0.65, 1.65]			
	width of arms	[1.0, 2.0]			
	width of legs	[1.0, 1.6]			
Dodge	angle	[-1.5, 1.5]	-	0.2M	1
Jump	energy	[0.8, 1.8]	foot contact	1.65M	0.92
	gravity	[0.2, 1.65]			
ObstacleJump	height of obstacle	[0.9, 3.0]	-	0.5M	1
Pivot	rotation	[0.8, 2.5]	-	0.12M	1
Punch	angle	[0.0, 1.0]	-	4.2M	0.72
	distance	[0.8, 1.4]			
	force	[1.0, 5.5]			
	height	[0.9, 1.1]			
Push	mass of object	[0.5, 10]	distance	0.7M	1
SideKick	angle	[0.0, 0.8]	-	1.52M	0.81
	force	[1.0, 6.0]			
	kick height	[0.9, 1.1]			
SpinKick	kick height	[0.5, 1.5]	-	0.93M	1
	rotation	[0.5, 2.0]			
Swing	height of obstacle	[1.0, 1.55]	-	1.5M	1
WallJump	height of obstacle	[0.9, 2.0]	-	1M	1

the whole-body angular momentum is preserved while the body is in the air, the angular velocity and the moment of inertia of the whole-body are inversely proportional to each other. Using our learning algorithm, the simulated character learned to squeeze its body to reduce its moment of inertia and consequently increase the angular velocity. Motion parameterization maps angular velocity parameters to the degree of whole-body extension.

The Cartwheel example demonstrates task parameterization by body conditions such as limb lengths and weights. Specifically, we selected four parameters, the length and radius of the arms and the legs. The weight is proportional to the length and the square of the radius. The total rotation angle for the duration of Cartwheel is constrained to prevent over-rotation and under-rotation. The angular velocity of the learned cartwheel varies nonlinearly depending on the mass distribution on limbs.

In the Push motion data, the simulated character pushes forward a box weighing 18kg. The Push skill is parameterized by the mass of the box ranging from 9kg to 180kg. We force the hands to stick to the box when they are close enough to the box within a certain phase. The total moving distance is fixed as a constraint regardless of the box weight. As the mass increases, the simulated character moves slowly and leans more on the box.

Table 2. Comparison of ablated algorithms.

skills	parameters	A1	A2	A3
spinkick	kick height	[0.5, 1.5]	[0.5, 1.5]	[0.5, 1.5]
	rotation	[0.725, 1.3]	[0.5, 2.0]	[0.5, 2.0]
jump	gravity	[0.4, 1.65]	[0.3, 1.65]	[0.2, 1.65]

7.2 Ablations

In this section, we conduct ablation studies to demonstrate the effectiveness of our approach. Our algorithm is reduced to the DeepMimic algorithm [Peng et al. 2018] if motion parameterization is removed. So, the DeepMimic algorithm is considered as the baseline of comparison (A1). The second algorithm allows spatial variations in motion parameterization while the task timing is fixed (A2). Our algorithm allows both temporal and spatial variations in motion parameterization (A3).

We compared the three algorithms for Jump and SpinKick examples. To simplify the experiments, the kinetic energy parameter is fixed in the comparison. All parameters and reward weights were tuned for the best performance of the A1 algorithm.

$\Delta\phi$ was set to a constant for the A2 algorithm to disable time warping. We adopted a simple curriculum generation method that explores the task space in a near-to-far order while alternating exploration and refinement phases of fixed duration. We measured

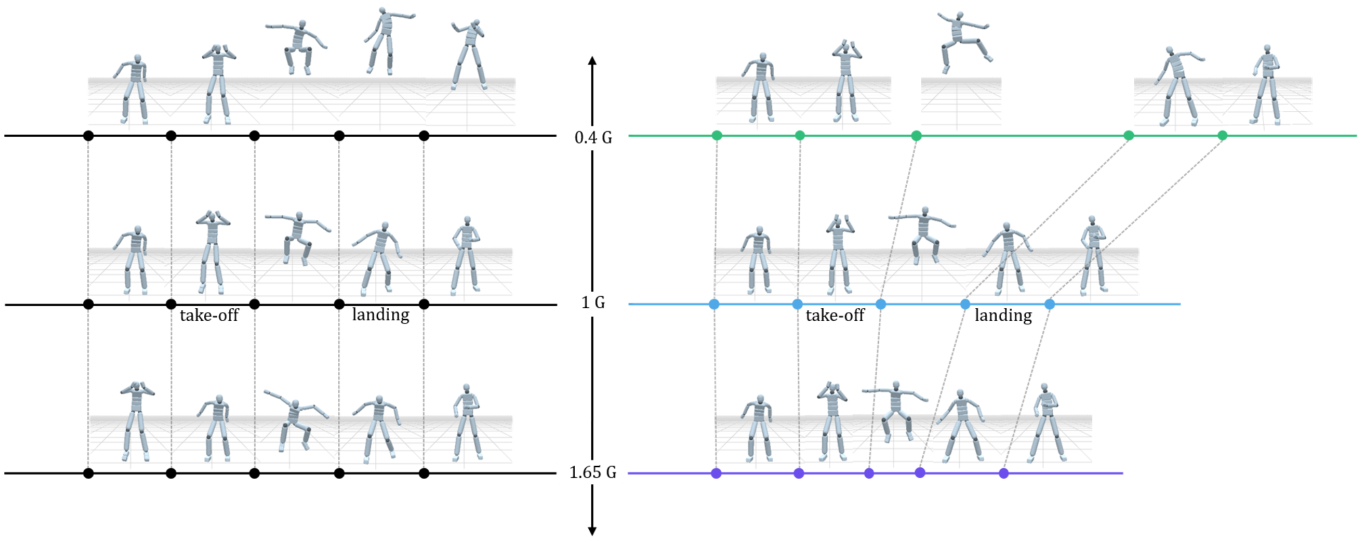

Fig. 6. The snapshots on timeline of jumping in different gravity settings with (left) A2 and (right) A3 algorithms. The dotted lines align the same phases in the jump motion. The time warping graph for each timeline is represented by a line of the same color in Figure 7.

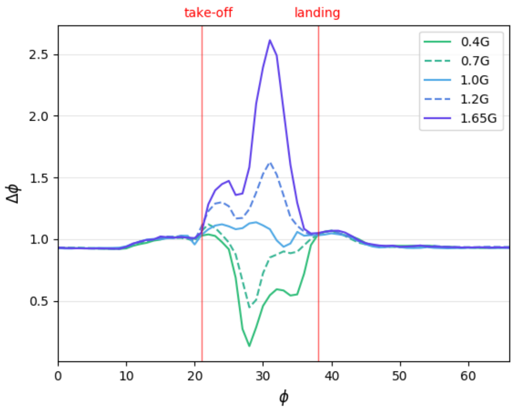

Fig. 7. The reference time and delta graphs for jumps in different gravity settings.

the ratio of successfully explored regions in the task space. The learning algorithm terminates when the exploration rate goes down to zero and the mean marginal value plateaus.

As expected, our algorithm (A3) successfully explored wider ranges in task spaces than the other algorithms (see Table 2). For the SpinKick example, it is relatively easy to change kick height, thus even the baseline algorithm explored the full range of kick heights. However, The baseline algorithm is not suitable to deal with large changes in full-body dynamics such as adjusting spinning angle and jump height. The comparison shows that our algorithm better deals with large changes in full-body dynamics. Our algorithm manifests exaggeration of anticipation and follow-through effects before and after the jump. The simulation results are best viewed in the accompanied video.

Figure 6 demonstrates the effectiveness of timewarping. The character is supposed to jump higher in the low gravity setting (0.4G). Although both A2 (no timewarp) and A3 (ours) algorithms generate higher jumps as expected, the result of A3 looks much better than the result of A2 because the A2 simulation does not align with the reference motion. In the figure (top, left), the pose in the middle of jumping matches the after-landing pose in the reference motion. This misalignment is noticeable in the simulation. Figure 7 shows the timelines for the range of gravity settings. Our algorithm learned to speed-up or slow-down appropriately to deal with gravity changes.

7.3 Comparison of Curriculum Methods

In this section, we compare our curriculum generation method to other methods to demonstrate its effectiveness. The baseline algorithm draws task parameters randomly from the task space without exploration and refinement phases. The second algorithm explores the entire task space first (ER=1) and moves on to the refinement phase. The third algorithm masters motor skills at heavily visited regions and proceeds to the next exploration phase. The mastery of motor skills is determined based on the threshold of mean marginal values. We also set a time limit at each phase so that the algorithm can move on. The mastering algorithm could be extremely slow without timeout. The fourth algorithm exploits the alternation between exploration and refinement phases and draws a constant amount of samples (20 value/policy updates) at each phase. Our algorithm uses the transition rules explained in Section 6.

We conducted experiments with four examples: SpinKick, Dodge, Jump and Push (see Figure 8). The SpinKick and Dodge are easier than Jump and Push examples. The ranges of task space are the same as Table 1 except for Jump. We fixed the kinetic energy parameter to simplify the experiment. The baseline algorithm expands quickly in the beginning for the easy examples but slows down as the probability of exploring heavily visited regions increases. The exploration-only algorithm performs well for the easy examples because intermittent refinement is not essential for robust exploration. Even though the exploration-only algorithm reaches ER=1 first in Spinkick example, it is not the fastest in the group because it has to spend some time in the refinement phase. The aggressive algorithms (baseline and exploration-only) often fail to explore the task space for the harder examples. For example, the baseline algorithm explored only 43% of the task space for Jump on average, and the exploration-only algorithm explored 87% for Jump and 80% for Push. On the other hand, the conservative algorithms (alternating and mastering) perform well for the challenging examples without being stuck in premature convergence. The refinement phases play an important role in incrementally refining motor skills and making the exploration phases gradually expand from the well-refined heavily visited regions. Our algorithm performs well regardless of the difficulty level and the type of examples because it seeks a good balance between performance and quality while being consistently successful in exploration.

7.4 Joint Torque Limits

In our experiments, setting reasonable joint torque limits is essential to construct believable motion parameterization (see Table 3). The character simulated without torque limits use superhuman forces to achieve goals or tasks in unrealistic ways. The superhuman character does not need to make bigger moves to generate more power. Motion parameterization can learn interesting, dynamically-realistic variants of the reference motion when the character has human physical abilities.

We conducted experiments to demonstrate the effects of joint limits on two motor skills: Punch and Push. We set joint torque limits heuristically. When the torque computed by a PD servo is beyond its limit, the computed torque is clipped by the limit. Figure 9 shows the simulation of the reference motion at the top row. The superhuman character in the second row can punch harder and push the heavier box without changing its moves. The motion of the superhuman character is almost identical to the reference motion. Our character with limited power learned that it could make a stronger punch impact when it takes a wider stance and a longer swing path. It also learned to lean more to push heavier boxes.

Table 3. The range of joint torque.

Joint	Torque range ($N \cdot m$)
Hips	[0, 0]
Spine	[-300, 300]
Spine1, Spine2	[-150, 150]
Neck, Head	[-75, 75]
Shoulders, Arms	[-150, 150]
ForeArms, Hands	[-90, 90]
UpLegs	[-300, 300]
Legs	[-225, 225]
Feet	[-145, 145]
Toes	[-90, 90]

8 DISCUSSION

We presented a new learning-based framework to construct a rich variety of parameterized motor skills learned from motion capture data. The key to the success of our approach is learning motor policies and motion parameterization concurrently. The concurrent learning approach achieves both computational efficiency and coherent visual quality over a wide range of the parameter domain. The databases of parameterized human motions are readily available to the public [Adobe Systems Incs. 2018]. We envision that the databases of parameterized motor skills for physics simulation will be available soon to the public as well.

Our algorithm is particularly effective when the reference motion is highly dynamic and energetic because dynamics effects are easily visualized, manipulated, and amplified in the motion parameterization. An interesting observation is that the effectiveness of exploration along the energy/dynamics axis is not symmetric. Exploration in one direction may be rapid, but not in the opposite direction. The algorithm easily deals with motor skills in heavier, faster, and stronger settings than the reference motion, while it sometimes struggles in lighter, slower, and weaker settings. The rationale is not clear yet. Perhaps, infusing more energy happens naturally as required by parameter settings, but reducing the level of energy requires more than parameter settings such as energy minimization/regularization.

There are many exciting directions to explore in future work. Currently, parameterized motor skills are learned for individual motion clips. It requires extra efforts to allow transitioning between motor skills that are learned independently from each other. Ideally, we wish to learn multiple, integrated motor skills simultaneously from unorganized motion datasets [Park et al. 2019].

Although our system runs successfully with various task types (ranging from static to highly energetic) and various parameter types (including body/physics/environment conditions), the current system implementation does not scale well with the dimension of the task space. We have tested up to four dimensions so far. Dealing with massively-parameterized motor skills will be an interesting challenge. The complexity of the real world problem requires many parameters for accurate modeling. Musculoskeletal gait is a good example [Lee et al. 2019]. There are many anatomical elements in the human musculoskeletal system (such as bones and muscles), and the condition of each element (such as bone length and muscle strength) affects gait. In this case, the dimension of the task space is proportional to the number of bones and muscles, which is close to 1000. Scalable algorithms would open up new possibilities in many practical applications.

ACKNOWLEDGMENTS

This work was supported by NCSOFT AI Center under Project Number NCSOFT2021-0767-K, and Institute of Information communications Technology Planning Evaluation(IITP) grant funded

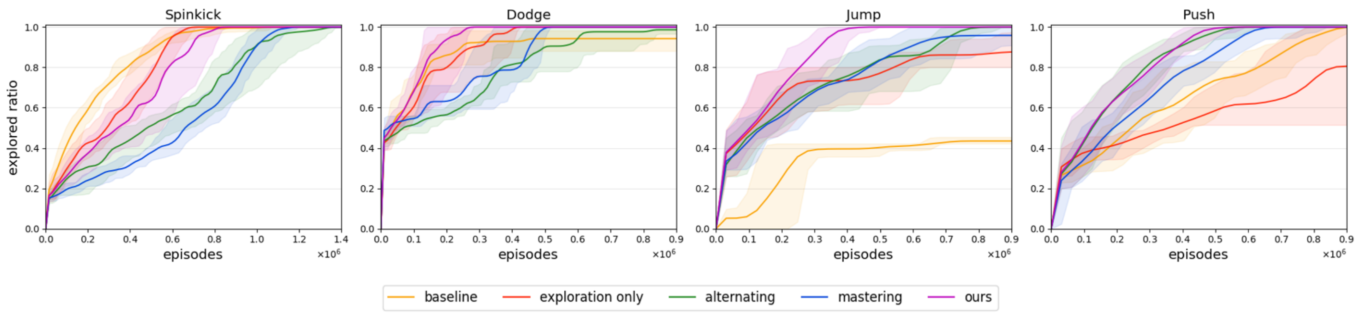

Fig. 8. The explored ratio in the task space for four different motor skills. The colored line is the average value of multiple attempts. The lower border remaining the same over time means one or more attempts got stuck in premature convergence.

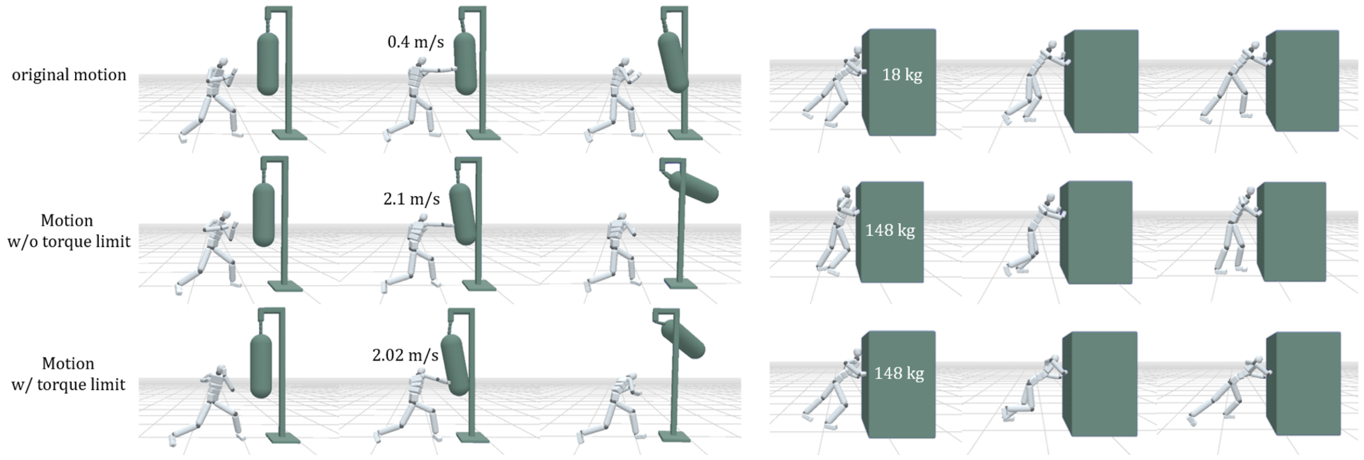

Fig. 9. The snaps of motor skills with/without torque limits. The first row is the simulation results tracking original reference, the second row is the simulation results without joint limit in harder/heavier settings, and the third row is the simulation results with joint limit in harder/heavier settings. The maximum velocity of the object after a stroke and the mass of the object are written on the figure.

by the Korea government(MSIT) (No.2017-0-00878, (SW Starlab) Human motion simulation based on deep learning).

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

Yeuhi Abe, C Karen Liu, and Zoran Popović. 2004. Momentum-based parameterization of dynamic character motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 173–182.

Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020a. Skeleton-Aware Networks for Deep Motion Retargeting. *ACM Transactions on Graphics* 39, 4 (2020), 62.

Kfir Aberman, Yijia Weng, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. 2020b. Unpaired Motion Style Transfer from Video to Animation. *ACM Transactions on Graphics* 39, 4 (2020), 64.

Adobe Systems Incs. 2018. *Mixamo*. https://www.mixamo.com

Shailen Agrawal, Shuo Shen, and Michiel van de Panne. 2014. Diverse motions and character shapes for simulated skills. *IEEE transactions on visualization and computer graphics* 20, 10 (2014).

Shailen Agrawal and Michiel van de Panne. 2016. Task-based Locomotion. *ACM Transactions on Graphics* 35, 4 (2016).

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*. 5048–5058.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.

Kevin Bergamin, Simon Claver, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Transactions on Graphics* 38, 6, Article 1 (2019).

Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. 2015. Computational Design of Walking Automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 93–100.

Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. 2018. Learning to Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning. *ACM Transactions on Graphics* 37, 6, Article 179 (2018).

CMU. 2002. *CMU Graphics Lab Motion Capture Database*. http://mocap.cs.cmu.edu

Michael Gleicher. 1998. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 33–42.

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-Based Inverse Kinematics. *ACM Transactions on Graphics* 23, 3 (2004), 522–531.

Perttu Hämäläinen, Perttu, Joose Rajamäki, and C. Karen Liu. 2015. Online Control of Simulated Humanoids Using Particle Belief Propagation. *ACM Transactions on Graphics* 34, 4, Article 81 (2015).

Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. 2008. Real-time motion retargeting to highly varied user-created morphologies. *ACM Transactions on Graphics* 27, 3 (2008), 1–11.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, Martin Riedmiller, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).

Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. 2010. Spatial Relationship Preserving Character Motion Adaptation. *ACM Transactions on Graphics* 29, 4, Article 33 (2010).

Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Transactions on Graphics* 39, 4, Article 53 (2020).

Seokpyo Hong, Daseong Han, Kyungmin Cho, Joseph S. Shin, and Junyong Noh. 2019. Physics-Based Full-Body Soccer Motion Control for Dribbling and Shooting. *ACM Transactions on Graphics* 38, 4, Article 74 (2019).

Yifeng Jiang, Tom Van Wouwe, Friedl De Groote, and C. Karen Liu. 2019. Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation. *ACM Transactions on Graphics* 38, 4, Article 72 (2019).

Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. 2009. Synchronized multi-character motion editing. *ACM transactions on graphics* 28, 3, Article 79 (2009).

Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. 2018a. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software* 3, 22 (2018).

Jehee Lee and Sung Yong Shin. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 39–48.

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018b. Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics* 37, 6, Article 180 (2018).

Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. 2006. Motion Patches: Building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics* 25, 3 (2006), 898–906.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable Muscle-actuated Human Simulation and Control. *ACM Transactions on Graphics* 38, 4, Article 73 (2019).

Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *International Conference on Machine Learning*. 1–9.

Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous Character Control with Low-Dimensional Embeddings. *ACM Transactions on Graphics* 31, 4, Article 28 (2012).

C. Karen Liu, Aaron Hertzmann, and Zoran Popović. 2005. Learning Physics-Based Motion Style with Nonlinear Inverse Optimization. *ACM Transactions on Graphics* 24, 3 (2005), 1071–1081.

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving Sampling-Based Motion Control. *Computer Graphics Forum* 34, 2 (2015), 415–423.

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* 31, 6, Article 154 (2012).

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-Based Contact-Rich Motion Control. *ACM Transactions on Graphics* 29, 4, Article 128 (2010).

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. 2020. CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion. *ACM Transactions on Graphics* 39, 4, Article 38 (2020).

Anna Majkowska and Petros Faloutsos. 2007. Flipping with physics: motion editing for acrobatics. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 35–44.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2019. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems* (2019).

James McCann, Nancy S Pollard, and Siddartha S Srinavisa. 2006. Physics-Based Motion Retiming. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 205–214.

Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. 2020. Catch amp; Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks. *ACM Transactions on Graphics* 39, 4, Article 39 (2020), 14 pages.

Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Transactions on Graphics* 38, 6, Article 208 (2019).

Igor Mordatch and Emo Todorov. 2014. Combining the benefits of function approximation and trajectory optimization.. In *Robotics: Science and Systems*.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of Complex Behaviors through Contact-Invariant Optimization. *ACM Transactions on Graphics* 31, 4, Article 43 (2012).

Tomohiko Mukai and Shigeru Kuriyama. 2005. Geostatistical Motion Interpolation. *ACM Transactions on Graphics* 24, 3 (2005), 1062–1070.

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and J. Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics* 38, 6, Article 205 (2019).

Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. 2002. On-Line Locomotion Generation Based on Motion Blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 105–111.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics* 37, 4, Article 143 (2018).

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Transactions on Graphics* 36, 4, Article 41 (2017).

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. 2019. MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. In *Advances in Neural Information Processing Systems*. 3686–3697.

Charles Rose, Michael F Cohen, and Bobby Bodenheimer. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications* 18, 5 (1998), 32–40.

Kwang Won Sok, Katsu Yamane, Jehee Lee, and Jessica Hodgins. 2010. Editing dynamic human motions via momentum and force. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer animation*. 11–20.

Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Transactions on Graphics* 39, 4, Article 54 (2020).

Jack M Wang, David J Fleet, and Aaron Hertzmann. 2007. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence* 30, 2 (2007), 283–298.

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Transactions on Graphics* 39, 4, Article 33 (2020).

Jungdam Won and Jehee Lee. 2019. Learning body shape variation in physics-based characters. *ACM Transactions on Graphics* 38, 6, Article 207 (2019).

Jungdam Won, Kyungho Lee, Carol O'Sullivan, Jessica K Hodgins, and Jehee Lee. 2014. Generating and ranking diverse multi-character interactions. *ACM Transactions on Graphics* 33, 6, Article 219 (2014).

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-Guided Control of Flapping Flight. *ACM Transactions on Graphics* 36, 6, Article 198 (2017).

Jungdam Won, Jungnam Park, and Jehee Lee. 2018. Aerobatics control of flying creatures via self-regulated learning. *ACM Transactions on Graphics* 37, 6, Article 181 (2018).

Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. 2020. ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Yuting Ye and C. Karen Liu. 2012. Synthesis of Detailed Hand Manipulations Using Contact Sampling. *ACM Transactions on Graphics* 31, 4, Article 41 (2012).

Zhiqi Yin and KangKang Yin. 2020. Linear Time Stable PD Controllers for Physics-Based Character Animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 191–200.

Ri Yu, Hwangpil Park, and Jehee Lee. 2019. Figure Skating Simulation from Video. *Computer Graphics Forum* 38, 7 (2019), 225–234.

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetric and Low-Energy Locomotion. *ACM Transactions on Graphics* 37, 4, Article 144 (2018).

M Ersin Yumer and Niloy J Mitra. 2016. Spectral Style Transfer for Human Motion between Independent Actions. *ACM Transactions on Graphics* 35, 4 (2016).