Physics-based Motion Capture Imitation with Deep Reinforcement Learning

Nuttapong Chentanez
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
Bangkok, Thailand
NVIDIA Research
Santa Clara, CA
nuttapong26@gmail.com

Matthias Müller
NVIDIA Research
Santa Clara, CA
matthias@mueller-fischer.com

Miles Macklin
NVIDIA Research
Santa Clara, CA
mmacklin@nvidia.com

Viktor Makoviychuk
NVIDIA Research
Santa Clara, CA
vmakoviychuk@nvidia.com

Stefan Jeschke
NVIDIA Research
Santa Clara, CA
sjeschke@nvidia.com

ABSTRACT

We introduce a deep reinforcement learning method that learns to control articulated humanoid bodies to imitate given target motions closely when simulated in a physics simulator. The target motion, which may not have been seen by the agent and can be noisy, is supplied at runtime. Our method can recover balance from moderate external disturbances and keep imitating the target motion. When subjected to large disturbances that cause the humanoid to fall down, our method can control the character to get up and recover to track the motion. Our method is trained to imitate the mocap clips from the CMU motion capture database and a number of other publicly available databases. We use a state-of-the-art deep reinforcement learning algorithm to learn to dynamically control the gain of PD controllers, whose target angles are derived from the mocap clip and to apply corrective torques with the goal of imitating the provided motion clip as closely as possible. Both the simulation and the learning algorithms are parallelized and run on the GPU. We demonstrate that the proposed method can control the character to imitate a wide variety of motions such as running, walking, dancing, jumping, kicking, punching, standing up, and so on.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; **Motion capture**; **Physical simulation**;

KEYWORDS

Mocap, Character Controller, Neural Network, Deep Learning, Simulation

ACM Reference Format:
Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-based Motion Capture Imitation with Deep Reinforcement Learning. In *MIG '18: Motion, Interaction and Games (MIG '18), November 8–10, 2018, Limassol, Cyprus.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3274247.3274506

1 INTRODUCTION

In this paper we tackle the problem of underactuated physically-based character control, where momentum can be changed only through contacts. Over the last two decades, numerous proposed methods have considerably improved the state of the art. An important issue in character control is to keep a high quality of natural motions on the one hand, while being able to reproduce a large variety of motions and motion styles on the other hand. For many existing methods that rely on heuristics like foot placement, Jacobian transpose strategy, base of support and/or hard-coded balancing rules, these two requirements are hard to fulfill simultaneously. The desire for being robust against external disturbances even complicates this situation. In this paper we are proposing a method that is able to imitate a given motion clip closely while at the same time allowing a large variety of motions. Motion clip-based character steering is a very common technique in computer games today. Technically we base our method on simple PD controllers with dynamically controlled gains together with corrective joint torques. The target angles for the PD controllers are directly derived from the provided motion clip at each time step. The key idea is now to compute the respective gains and the corrective torques using a deep reinforcement learning agent. We trained this agent with a large database of mocap examples without any additional heuristics. The root is not actuated and our controller conserves momentums. Note that employing the agent alone to steer the PD controller torque and/or target angle without using the target angle from the mocap clip would make the control problem very difficult. This

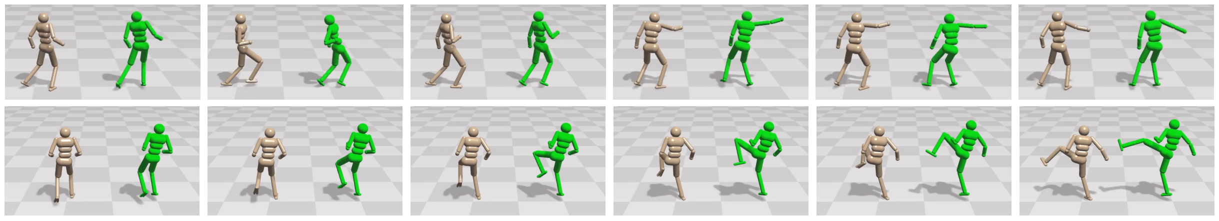
Figure 1: Punching and kicking mocap clips imitated by our agent.

would effectively limit the variations in motion and style that can faithfully be reproduced. Previous work on imitating a given motion clip [Merel et al. 2017; Peng et al. 2018, 2017; Peng and van de Panne 2017] has used this approach, and their agent can only imitate one or very few motion clips per trained network. By contrast, our combined method can imitate clips sampled from a mocap database with thousands of motions and even clips that are not seen during the training with a single neural network. We are not aware of any previous approach based on deep reinforcement learning that is able to reproduce such a large motion variety. In addition, the network training is an ongoing process, meaning that the variety of reproducible motions can be improved with new examples and more training.

Also note that using the PD controller alone without corrective joint torques would practically always result in the agent to crawl about the floor, as it fails to balance and does not properly apply torque to result in the correct contact forces required to produce a desired motion. By contrast, our combined approach even allows the character to react to a dynamic environment to some extent. For example, we can randomly throw objects at a character to disturb its motion without having it immediately falling down. For the case that it falls after a major disturbance, for example, we use a special *recovery agent* that attempts to get the character back close to the desired pose in the mocap clip. In summary our main contributions are:

- A novel action space that consists of both the joint torque and the gain of PD controllers.
- A state representation, a reward function, a state distribution and a schedule for training the agent with deep reinforcement learning.
- A decomposition of the controller into tracking and recovering phases to allow the controller to be able to imitate the mocap clip while being able to recover from large disturbances that let it fall, for example.
- GPU parallel rollout with dead agent being reset immediately to avoid stalling.
- GPU training with Proximal Policy Optimization (PPO).

2 RELATED WORK

Various previous work employed physics-based character controllers which exert torque directly or indirectly to joints of articulated rigid bodies in a physics engine to produce realistic animations. Early works utilizing Finite State Machines (FSMs) with feedback control include [Coros et al. 2010; Hodgins et al. 1995; Laszlo et al. 1996; Lee et al. 2010; Sok et al. 2007; Yin et al. 2007]. Evolution-based methods have been proposed to optimize controllers for linked blocks characters [Sims 1994], swimming characters [Grzeszczuk and Terzopoulos 1995; Tan et al. 2011], bipeds [Wang et al. 2009, 2010], and bipeds with muscles [Geijtenbeek et al. 2013; Wang et al. 2012]. Many of these methods utilize Covariance Matrix Adaptation (CMA) [Hansen 2007]. Trajectory optimization has been used for physics-based character control in [Al Borno et al. 2013; Levine and Abbeel 2014; Levine and Koltun 2014; Mordatch et al. 2015; Mordatch and Todorov 2014; Mordatch et al. 2013]. We refer the reader to the excellent survey [Geijtenbeek and Pronost 2012] for more approaches in physics-based character controllers and focus on related work based on reinforcement learning and motion clip tracking below.

Reinforcement learning (RL) has been used by a number of previous works for controlling physically simulated characters. Many approaches utilize RL to select and/or provide input to FSM controllers. Colos et al. [Coros et al. 2009] uses RL to train a policy to select among FSM controllers to achieve a goal. Peng et al. [Peng et al. 2015] uses RL to select among FSM controllers for each locomotion cycle to achieve locomotion over terrains with gaps and obstacles. Peng et al. [Peng et al. 2016] uses a CACLA RL algorithm [van Hasselt and Wiering 2007] with a neural network as a function approximator to train a policy for FSM controller selection as well as for continuous FSM control parameters to achieve locomotion over sloped terrain for various characters.

More recent approaches utilize RL to directly control the agent without FSM controllers. Lillicrap et al. [Lillicrap et al. 2015] demonstrated learning locomotion for 2D articulated characters controlled with joint torque using a Deterministic Policy Gradient (DDPG) algorithm. Schulman et al. [Schulman et al. 2015a] achieve similar results using a Trust Region Policy Optimization (TRPO) algorithm. They later improve it by utilizing generalized advantage estimation and show that they can learn locomotion for 3D humanoids [Schulman et al. 2015b]. Peng et al. [Peng and van de Panne 2017] investigate whether torque, target angle, or target velocity control are the best for tracking a single mocap clip. Peng et al. [Peng et al. 2017] train two controller levels with RL for locomotion and ball dribbling, where the low-level controller goal is to match provided footsteps and the high-level controller places the footsteps to follow some user input and to avoid obstacles. Merel et al. [Merel et al. 2017] use generative adversarial imitation learning (GAIL) to train an agent to mimic a single or few motion clips. Supervised learning is used for training a discriminator to distinguish the mocap clip from the motion produced by RL while RL gets positive reward by fooling the discriminator. Pure RL with a simple reward function is used for training humanoid locomotion in a diverse set of environments in [Heess et al. 2017] with gradually increasing difficulty, resulting in a natural looking motion. Glen et al. [Berseth et al. 2018] combine various policies representing locomotion skills and used transfer learning to incrementally add new skills. Most recently, Peng et al. [Peng et al. 2018] demonstrates varieties of impressive skills learned with neural networks. The agent receives both the task based reward and motion matching reward. They reported that a single network can be trained to imitate up to a handful of motions. None of these RL works is trained on a large number of mocap clips with diverse types of motions like we do in this work.

A number of approaches aimed to produce a controller that tracks mocap clips. Sok et al. [Sok et al. 2007] and Yin et al. [Yin et al. 2007] performed offline optimization to generate a closed-loop controller that tracks a running mocap clip. Lee et al. [Lee et al. 2010] track motion clips provided during run time using various feedback control rules to provide balance. They dynamically query the appropriate motion clip and make adjustments to the clip on the fly. Kwon and Hodgins [Kwon and Hodgins 2010] use a PD controller to track a motion clip and use an inverted pendulum model to approximate the torque needed to provide balance. Cooper and Ballard [Cooper and Ballard 2012] use fixed joints and forward simulation to compute torque needed to achieve the same target angles as motion capture data and modify foot placement to provide balance. Geijtenbeek et al. [Geijtenbeek et al. 2012] use PD controllers to track motion and Jacobian transpose control to provide balance. The parameters for the controllers are optimized with CMA for each individual motion clip. Wang et al. [Wang et al. 2014] use trajectory optimization to optimize for joint torque that produces motion similar to a provided mocap clip. Liu et al. [Liu et al. 2016] use SAMCON [Liu et al. 2015, 2010], a sample-based approach for generating controllers for short segments of motion clips and they generate a control graph that allows transitions between motions and provides a way to select a new appropriate controller when the character drifts far from the reference motion. Kavafoglu et al. [Kavafoglu et al. 2018] use CMA to optimize for parameters to provide balance for various walking motion clips, given velocity deviation. Our work differs from these previous works in that we attempt to learn a single controller that can imitate a wide variety of motions. This controller attempts to imitate an arbitrary reference motion clip at runtime, which can be one of many types of motion and has potentially not even been seen by the agent before. We do not provide any heuristic to the learning agent like foot placement, Jacobian transpose strategy, base of support, and so on. The agent also has to learn not only to balance but also to exert extra control torques and dynamically adjust the gain of PD controllers to try to match the resulting motion better to the provided motion clips.

3 METHOD

In this section we will describe how we train the *tracking agent* to imitate a provided motion capture clip and *recovering* agent to recover from large external disturbance. First, we will explain how we prepare the motion data in 3.1. We then outline our character controller in 3.2 and describe how we train the agents in 3.3. Finally we show how the learned agents can be utilized at runtime in 3.4. Unless otherwise stated, our units for distance, mass and angles are meter, kg, and radians respectively. The specific values of the parameters we used in our experiments can be found in Table 1 and Table 2. Throughout this section, we define the following functions: QuatToEuler() for converting quaternion to Euler angles, EulerToMatrix33() for converting Euler angles to 3x3 rotation matrix, Matrix33ToQuat() for converting a 3x3 rotation matrix to quaternion, RotateInv() for rotating a vector with an inverse of a quaternion, Average() for computing the average of a list, Count() for counting the number of elements in which the value is true, RMS() for computing the root mean square of a list.

3.1 Motion Capture Data Preparation

We employ a 29 DOF humanoid articulated Mujoco model [Kumar [n. d.]] with modified limbs' length to match with those used in [Holden et al. 2016] and radii modified to match an average human. We use the motion capture database from [Daniel Holden [n. d.]] which combined the CMU motion capture database [cmu [n. d.]] with various other publicly available databases along with their own captures. The poses of head, torso, upper arms, lower arms, hips, upper legs, lower legs, and feet are used for creating fixed joints to pull the humanoid articulated model. For each frame of every mocap clip, we obtain the poses and set the fixed joints. Then we run our rigid body physics simulation to solve for the rigid body poses that closely follow the mocap, where collisions between individual bodies are disabled. We use a GPU based rigid body simulation [anonymized for review 2018] and set the compliance of all fixed joints to 10^{-4}. This step is essentially an Inverse Kinematics (IK) solver. All resulting body poses are stored for later use. The linear and angular velocities are also calculated and stored for each body by using finite differencing of the poses in two consecutive frames.

3.2 Character Controller

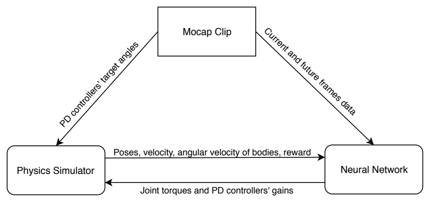
Mocap Clip
PD controllers' target angles
Current and future frames data
Poses, velocity, angular velocity of bodies, reward
Physics Simulator
Neural Network
Joint torques and PD controllers' gains

Figure 2: Information flow between physics simulator, the provided mocap clip and the neural network.

The humanoid is controlled by a reinforcement learning agent that has the goal to imitate the provided motion clip. To be more precise, the agent's observation consists of the current state of the humanoid, the poses of all the body parts in the current frame and a few frames in the future. Based on the observation, the agent produces an action which controls the humanoid such that the resulting motion is as similar as possible to the provided motion clip. For our case, the action consists of the joint torques and the PD controller gains. The exact observation the agent receives will be discussed in great detail later on in 3.3. In this section we first focus on the control side. Figure 2 shows the flow of information between physics simulator, the provided mocap clip and the neural network.

Let p_i, q_i, v_i, ω_i where $1 \le i \le n$ be the position, orientation, velocity and angular velocity of the rigid bodies of the character respectively, where $i = 1$ indicates the torso and n is the number of rigid bodies. Let a_j, a_j^{low}, a_j^{high} a_j^{vel} where $1 \le j \le t$ be the joint angles, lower joint angle limits, upper joint angle limits and joint angular velocity respectively where t is the number of joints. Let pow_j be the power of the motor at each joint specified in the MJCF file. Let f_i be the sum of all contact forces on the i^{th} body, and f_i^c be a binary value to indicate if the i^{th} body contacts with anything other than the ground. Let p_i^m, q_i^m, v_i^m, ω_i^m be the quantities for the provided mocap clip at the m^{th} frame in the future from the current frame. We define z to point upwards in our world coordinate system, meaning that gravity acts in the -z direction. Let

$$\theta_{roll}, \theta_{pitch}, \theta_{yaw} = QuatToEuler(q_1), \tag{1}$$

, and let

$$M = EulerToMatrix33(0, 0, \theta_{yaw}) \tag{2}$$

which aligns the x-axis to the direction the torso is heading and $q^M = Matrix33ToQuat(M)$. We also define $e_{pos} = ||p_1 - p_1^0||_2$ and e_{rot} to be the magnitude of the smallest rotation angle to align q_1 with q_1^0. They represent the positional and rotational error of the torso of the simulated agent with the current frame of the mocap clip.

Our key idea is to use a combination of a target angle PD controller whose gain is dynamically controlled by the agent, with an additional joint torque, also provided by the agent to imitate the provided mocap clip. Specifically, for each joint, j, the target angle PD controller exert torque equal to

$$k_j^s(\theta_j^{target} - \theta_j) - k_j^d \dot{\theta}_j, \tag{3}$$

where θ_j^{target} is the target angle, θ_j and $\dot{\theta}_j$ are the joint angle and joint angular velocity respectively, k_j^s is the stiffness which is computed as described next. The agent produces actions c_j and s_j, where $-1 \le c_j^{prev}, c_j \le 1$ and $0 \le s_j \le 1$ at each time step. We set $k_j^s = s_j k_s pow_j$. The target angles, θ_j^{target}, are obtained from the body poses in the current frame of the provided motion clip. The damping coefficient used is k_d. The PD controllers are integrated implicitly by our rigid body solver. In addition, we apply joint torque $k_t c_j pow_j$ to each joint explicitly.

Essentially, the agent has to learn to produce control torque and the PD controller gain such that the resulting motion closely matches with the provided clip. The PD controller generally pulls the limbs toward the desired target angles, but the amount is controlled by the agent. This allows the agent to even turn off the PD controller, if it chooses to. We experimented with having the agent to directly set the target angles with fixed gain, and also to use torque only without any PD controller. If only a single mocap clip is used in the training these choices of action learn much slower than our proposed combination of PD and torque controllers. If numerous mocap clips from the database are used, these choices of action space produce significantly inferior motion quality both in terms of reward received and the visual quality. We experimented with a number of different reward functions and observe this consistently. Using torque on top of a fixed gain PD controller alone also doesn't produce as good quality result as having the gain of the PD controller being dynamically controlled by the agent. Figure 3 shows a plot of the reward using torque only, torque + fixed gain PD controller, and torque + variable gain PD controller (our method) for various cases. The last part of our accompanying video shows the result of torque only controller vs our method. Generally speaking, the torque only controller can globally match the center of mass position and orientation very well for most animations and hence it still can achieve relatively high reward, however, the style of the movement is lost. Using a fixed gain PD controller with torque correction generally produces motion quality similar to the variable gain PD controller, however, for some fast moving motion or noisy mocap clips, the agent can lose balance and fall down easily as it can't tune down/ignore the effect of the PD controller and hence achieve lower reward.

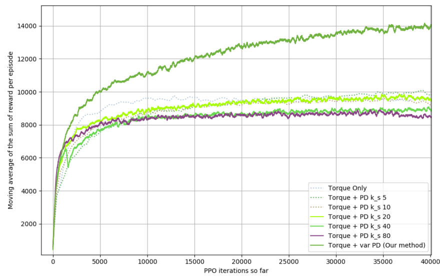

Figure 3: Sum of reward per episode obtained vs. PPO iteration so far of various controllers. Pure Torque, Torque with fixed gain PD controller of $k_s = 5, 10, 20, 40, 80$ and $s_j = 1$, and Torque with variable gain PD controller ($k_s = 20$ and $0 <= s_j <= 1$ output by the agent). Our variable torque + PD controller produces significantly higher reward than other choices. The sum of reward shown is a moving average over a window of 200 PPO iterations, to reduce high frequency noise.

3.3 Training

We use two different agents to control the character, where only one actively controls the character at a time. The first agent is called *tracking agent* with the goal to imitate the mocap clip. It is used when the character is closely imitating the mocap clip. The second agent is called *recovering agent*, and it is used when the character is far away from the mocap clip, for example when it falls down or when external disturbances are large. Its goal is to try to get the agent back to be close enough to the mocap clip so that the *tracking agent* can take over the control again. We use different observations for these two agents, aiming to make the information useful for the different tasks. During training, we also use different reward functions and different strategies for sampling the initial state, as their goals are different.

3.3.1 Tracking Agent. For the *tracking agent*, we extend upon the observation used by RoboSchool [Rob [n. d.]] to include more information and also the motion clip data. Specifically, we start with the observation from RoboSchool which contains

- $p_0.z$ • $\sin(\theta)$ • $\cos(\theta)$ • $M^{-1}v_1$ • $M^{-1}\omega_1$
- θ_{roll} • θ_{pitch} • a_j • $f^g_{leftfoot}$ • $f^g_{rightfoot}$

where $\theta = atan2(p^0_1.y - p_1.y, p^0_1.x - p_1.x) - \theta_{yaw}$. We then append the observation with

- c^{prev}_j • s^{prev}_j • $M^{-1}f_i$ • $\frac{t_{bad}}{t^{max}_{bad}}$ • $M^{-1}(p_i - p_1)$

and for each $m = \{0, 4, 16, 64\}$,

- $q^M \cdot q^m_1$ • $M^{-1}v^m_1$ • $M^{-1}\omega^m_1$ • $M^{-1}(p^m_i - p_i)$

where c^{prev}_j and s^{prev}_j are the agent's control output in the previous time step. t_{bad} is first initialized to 0. It gets incremented if $e_{pos} > e^{bad}_{pos}$ or $e_{rot} > e^{bad}_{rot}$. It gets decremented (and clamped to 0) if both errors do not exceed the thresholds. If the mocap frame in the future exceeds t_{max}, we simply pad it with the last mocap frame. We provide sparse and adaptive sampling of the future frames of the mocap clip because they provide the agent with short, medium and long term goals without too much potentially redundant data which can slow down the learning process. The reward for the *tracking agent* is

$$r_{electricity} + r_{joint@limit} + r_{feet\_collision} + r_{alive} + r_{pos} + r_{rot} + r_{vel} + r_{avel} + r_{local} + r_{height} + r_{contact} + r_{dead}$$

The terms $r_{electricity}$, $r_{joint@limit}$, and $r_{feet\_collision}$ are similar to RoboSchool:

$$r_{electricity} = k_{electricity} Average(|a^{vel}_j c_j pow_j / max_k(pow_k)|) + k_{stall\_torque} Average(c^2_j),$$

$$r_{joint@limit} = k_{joint@limit} Count(|\frac{(a_j - 0.5a^{high}_j + a^{low}_j)}{a^{high}_j - a^{low}_j}| > 0.495),$$

$$r_{feet\_collision} = k_{feet\_collision}((f^c_{right\_foot>0}) or (f^c_{left\_foot>0})).$$

We propose to use other terms in the following forms:

$$r_{alive} = k_{alive}, r_{pos} = k_{pos} max(0, \frac{e^{ok}_{pos} - e_{pos}}{e^{ok}_{pos}}),$$

$$r_{rot} = k_{rot} max(0, \frac{e^{ok}_{rot} - e_{rot}}{e^{ok}_{rot}}),$$

$$r_{vel} = k_{vel} max(0, \frac{e^{ok}_{vel} - e_{vel}}{e^{ok}_{vel}}), \text{where } e_{vel} = ||v_1 - v^0_1||_2,$$

$$r_{avel} = k_{avel} max(0, \frac{e^{ok}_{avel} - e_{avel}}{e^{ok}_{avel}}), \text{where } e_{avel} = ||\omega_1 - \omega^0_1||_2$$

$$r_{local} = k_{local} max(0, e^{ok}_{local} - RMS_{i'}(RotateInv(q_1, p_{i'} - p_1) - RotateInv(q^0_1, p^0_{i'} - p^0_1))/e^{ok}_{local},$$

where RMS (Root mean squared) is taken over some of the body parts indexed by i' as follows: torso1, right_upper_arm, right_hand, right_foot, left_upper_arm, left_hand and left_foot, which makes the agent prefer to move the limbs to match that of the mocap.

$$r_{height} = k_{height}(\frac{e^{ok}_z - max(0, p^0_1.z - p^1.z)}{e^{ok}_z})^2,$$

incentivizes the agent to achieve a height greater than or equal to that of the target. The following term

$$r_{contact} = k_{contact} \sum_{i'} ||f_{i'}||_2$$

penalizes excessive contact force, where the sum is computed for the same body parts as r_{local}. The final term

$$r_{dead} = k_{dead}(t_{bad} > t^{max}_{bad})$$

is to heavily penalize the agent if it falls down before the maximum time step is reached. We also terminate the agent when $t_{bad} > t^{max}_{bad}$ or if the time step reaches t_{max}. We choose to use a linear falloff for these reward terms instead of an exponential falloff as used in [Peng et al. 2017; Peng and van de Panne 2017] because we found that the agent learns slower with an exponential falloff in our preliminary tests. A potential explanation for this is when applying a low exponent coefficient for exponential decays, the reward an agent receives for a low error is not significantly higher than the reward for a moderate error.

We sample the mocap clip from the database by randomly choosing a clip and a starting frame and define the remaining frames to be the mocap clip the agent gets trained to imitate. If the number of frames is smaller than t_{max}, we randomly pick a frame from a clip that is similar to the last frame in the clip and append the frames transformed so that the torso's x and y coordinate and the yaw angle matches. We repeat the process until we have t_{max} frames. We pre-compute potential transitions by measuring the RMS positional errors of the bodies center of mass of the last frame of each clip against every frame in every other clip as was done in [Kovar et al. 2002]. However, we chose a coordinate system that aligns the two frames' torso yaw angles, instead of the least square fit frame. We allow transition to all those frames with an error < 0.1 or to the frame with a minimum error, if none of the frames has an error < 0.1.

To reduce irrelevant data, we remove clips from the database that are obviously not captured when the actor moved around on a flat floor. We detect such clips by counting the number of consecutive frames that both feet are not on the floor, i.e. where their z coordinate > 0.4. If this condition applied to more than 50 consecutive frames, we remove this clip from the database. Finally, we also remove some small number of clips from the database for the purpose of training. These removed clips are saved separately and are used for testing the ability of our agent to generalize to unseen clips.

We initialize the agent with the poses of the rigid body of the first frame of the sampled mocap clip as in [Merel et al. 2017]. We also experimented with adding small amounts of noise to the joint angles, linear velocity and angular velocity but found that this did not provide any significant change in learning speed and reward. We think that this is because there are already a lot of possible initial states (>2.5 Million frames) and the noise during the training already sufficiently jitters the agents to see sufficiently different states.

3.3.2 Recovering Agent. The goal of the *recovering agent* is to direct the character from its current state towards a target pose that is potentially very different. During this phase, the motion clip is paused and only its current frame is relevant. For the observation we start with that from Roboschool and append

- c_j^{prev} • s_j^{prev} • $M^{-1}f_i$ • $M^{-1}(p_i - p_1)$
- $q^M * q_1^0$ • $M^{-1}v_1^0$ • $M^{-1}\omega_1^0$ • $M^{-1}(p_i^0 - p_i)$.

Notice that we do not include information from future mocap frames in this case.

The reward function takes a similar form as for the *tracking agent* with some modifications. We increase k_{height}, k_{pos} and k_{rot} and zero out k_{vel}, k_{avel} as can be seen in Table 1. We also zero out r_{rot}, r_{local} if $e_{pos} > e_{pos}^{ok}$, essentially making the agent first care about matching the height and position. Only if the height and position do match reasonable well, the agent attempts to match the orientation and local pose. The velocity and angular velocities are ignored.

During training, we randomly pick a frame from the mocap database as the target pose. As for the initial pose, with 50 percent chance, we initialize the character with another random frame from the mocap database. Otherwise we initialize it with a pose lying on the floor in various manners. These lying poses are generated in a preprocessing step that initializes the agent using a frame from mocap and exert random torques on joints for several time steps, then let it fall to the ground in a ragdoll manner. After the body stopped moving, we save this pose as lying on the floor. Training the agent this way results in a *recovering agent* that can make the character successfully stand up from most random poses. We also use a much smaller t_{max} for the recovering agent, which forces the agent to more quickly get to the target pose.

3.3.3 Learning Algorithm. We train the agent with the parallel PPO algorithm [Schulman et al. 2017] using a modified version of OpenAI's Baselines [Dhariwal et al. 2017] PPO1 implementation which uses GPU accelerated Tensor Flow [Abadi et al. 2015]. Our policy and value neural networks keep track of the running mean and standard deviation of the input to normalize and clip it to be between -5 and 5. We use 4 hidden layers of 512 scaled exponential linear units [Klambauer et al. 2017] which we found to generally yield faster learning compared to the default tanh units. The standard deviation for the action sampling during training is fixed to 1 for all dimensions. We modify PPO to use an adaptive optimization step size, by controlling the target KL divergence between the old and the new policy. If the current KL divergence is more than two times the target KL divergence, we reduce the optimization step size by a factor of 1.5. If the current KL divergence is less than half of the target KL divergence, we increase the optimization step size by a factor of 1.5. This approach allows for reduced variance during the training process and allows for more aggressive updates when the KL divergence is small. We use a target KL divergence of 0.01 in all our experiments and initialize the optimization step size to 5×10^{-4}. We use the discount factor $\gamma = 0.99$ generalized advantage estimate discount factor $\lambda = 0.95$ and clipped surrogate objective $\epsilon = 0.2$. We point the reader to [Schulman et al. 2017] for an explanation of these parameters.

We simulate 500 agents in parallel using a GPU rigid body solver [anonymized for review 2018] to rollout the current policy. Hence, we modify Baselines to evaluate actions and state values of all agents in parallel and communicate with the simulator at every time step. Dead agents are individually reset immediately. We perform one PPO update every 200 time steps, which corresponds to 100k samples. Resetting dead agents immediately slightly skew the rollout to collect more short episodes compared to longer ones. This tends to make the reward vs. number of PPO updates (when the same number of samples is used per PPO update) grow slower compared to using an equivalent number of serial time steps, or stalling dead agents to not generate more samples and reset all the agents at the same time. However, with a large number of agents as in our case, stalling is not efficient and the slower growth of reward vs. number of PPO updates is more than made up for by having a much faster rollout. Most of the computation steps including the whole simulation and PPO are executed on the GPU.

We experimented with having only a single network for both tracking the input mocap and recovering from large disturbance by taking the union of states and combining the reward functions, by unionizing the terms and tweaking weights and stop the motion capture frame incremental when $e_{pos} > e_{pos}^{bad}$ and/or $e_{rot} > e_{rot}^{bad}$ to allow the agent to "catch up". Some of the agents are initialized in the same way as *tracking agent* and the rest in the same way as *recovering agent*. Despite a lot of trials we found the performance of both tracking and recovering to not come close to having separate networks for these tasks, so we keep them separate.

3.4 Runtime

During runtime we use the *tracking agent* to control the character as long as $e_{pos} < e_{pos}^{bad}$ and $e_{rot} < e_{rot}^{bad}$. Otherwise, we pause the mocap clip and switch to the *recovering agent*. We generally would like to use the *tracking agent* to match the input mocap clip, but if it fails for various reasons such as external disturbance, we use the *recovering agent* to get the character back to where the agent can continue to track ie. when $e_{pos} < e_{pos}^{bad}$ and $e_{rot} < e_{rot}^{bad}$. During run time, we always choose the greedy action output by the network.

Table 1: Parameter values used, common to *tracking agent* and *recovering agent*

Name	Value	Name	Value
k_s	20	$k_{joint@limit}$	-0.2
k_d	100	$k_{feet\_collision}$	-0.2
k_t	0.81	k_{alive}	1.5
e_{pos}^{ok}	0.6	e_{vel}^{ok}	2.0
e_{pos}^{bad}	1.0	e_{avel}^{ok}	2.0
e_{rot}^{ok}	$\frac{\pi}{2}$	e_{local}^{ok}	0.1
e_{rot}^{bad}	π	e_z^{ok}	1.0
t_{bad}^{max}	30	$k_{contact}$	$-\frac{1}{300}$
k_{dead}	-400	$k_{electricity}$	-7.2
k_{stall_torque}	-0.5		

Table 2: Parameter values for *tracking agent* and *recovering agent*

Name	*Tracking Agent*	*Recovering Agent*
k_{pos}	1.0	6.0
k_{rot}	2.0	3.0
k_{vel}	1.0	0.0
k_{avel}	1.0	0.0
k_{local}	9.0	0.6
k_{height}	4.0	6.0
t_{max}	2000	180

3.5 Results

We performed our experiments on a NVIDIA Tesla V100-SXM2-16GB GPU. For our *tracking agent*, we experimented with up to 7 hidden layers and found that 3 to 6 layers provide the best sum of reward per episode. We also experimented with 2 hidden layers with more units, but found that the reward received for the same number of network weights is lower than having more layers. The reward obtained for various cases are shown in Figure 4. The animations shown in the figures and the videos are generated with 4 hidden layers. Throughout this paper and the accompanying video, the simulated characters are colored in brown while the target mocap clip is colored in green. 40000 PPO iterations including the simulation and learning takes about 120 hours of computation times on a single GPU. The reward plot for our *recovery agent* is shown in Figure 5.

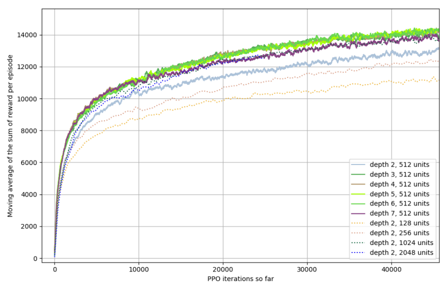

Figure 4: The sum of reward per episode vs number of PPO iterations so far of our *tracking agent* using different number of hidden layers and number of hidden units per layer. Two layer perform worse than three or more layers. Three to six layers yield a similar level of reward. Seven layers yields a smaller reward. The sum of reward shown is a moving average over a window of 200 PPO iterations, to reduce high frequency noise.

Figure 6 shows that our method can track walking, crouch walking and running mocap clips. Figure 1 shows punching and kicking mocap clips being tracked by our agent. Due to the random sampling nature of our training method and the fact that our database consists of over 10 millions frames, these exact clips may have never been seen or may have been seen only a few times by the agent. The results in our accompanying video demonstrate that our method is able to imitate a remarkably wide variety of motions and

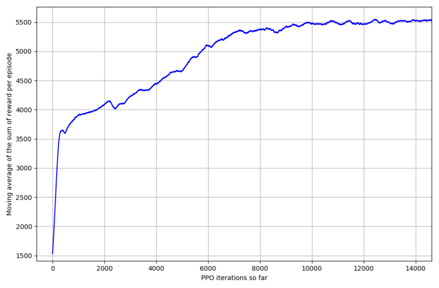

Figure 5: The sum of reward per episode vs number of PPO iterations so far of the recovery agent. The sum of reward shown is a moving average over a window of 200 PPO iterations, to reduce high frequency noise.

styles. Some of the motions also contain a significant amount of noise, as the mocap data is not cleaned up. Nonetheless, our agent can tolerate this noise to a certain extent. Figure 7 shows the agent imitating clips that are not used during training, demonstrating the ability of our agent to generalize to unseen clips. Figure 8 top row shows the agent being hit by a small box thrown by the user. The *tracking agent* is able to keep controlling the character. The bottom row shows the agent falling down after being hit by a big box. Here, the *recovering agent* is activated to bring the character back to a pose close to the current mocap clip frame.

We also experimenting with omitting some reward terms and train for 40000 PPO iterations. The accompanying video shows that omitting r_{local}, which causes the agent limbs to not match mocap, and r_{pos} which results in the global position to not follow the mocap.

4 DISCUSSION

We proposed a method for imitating a provided motion clip with a physics-based controller using reinforcement learning. The controller can tolerate some amount of perturbations. When it is subjected to large perturbations or when it fails to track the provided motion and the character falls down, it attempts to get the character up again and to come back to track the motion. Figure 10 shows the frame reached before the *tracking agent* is dead, starting from the first frame, for each padded mocap clip from the database averaging over 20 trials. The animations are sorted from the highest to the smallest frames reached. 1382 clips can be tracked from start to end and 3200 clips can be tracked for more than half their lengths. 208 clips failed to be tracked for more than 50 frames, most of which are the motions on non-flat terrain that are not pruned out by our detection heuristics.

While our method can track a wide variety of provided mocap clips, it still can fail to track certain motions such as some flips, some fast jumps, and some break dances, examples of which are shown in Figure 9 and in the accompanying video. For these motions it is tricky to keep the balance. Nonetheless, we experimented with training a dedicated network to imitate only one and a few of these motions and found that the network can imitate them faithfully. This suggests that the network architecture is capable of representing the policy that successfully imitates these motions. It

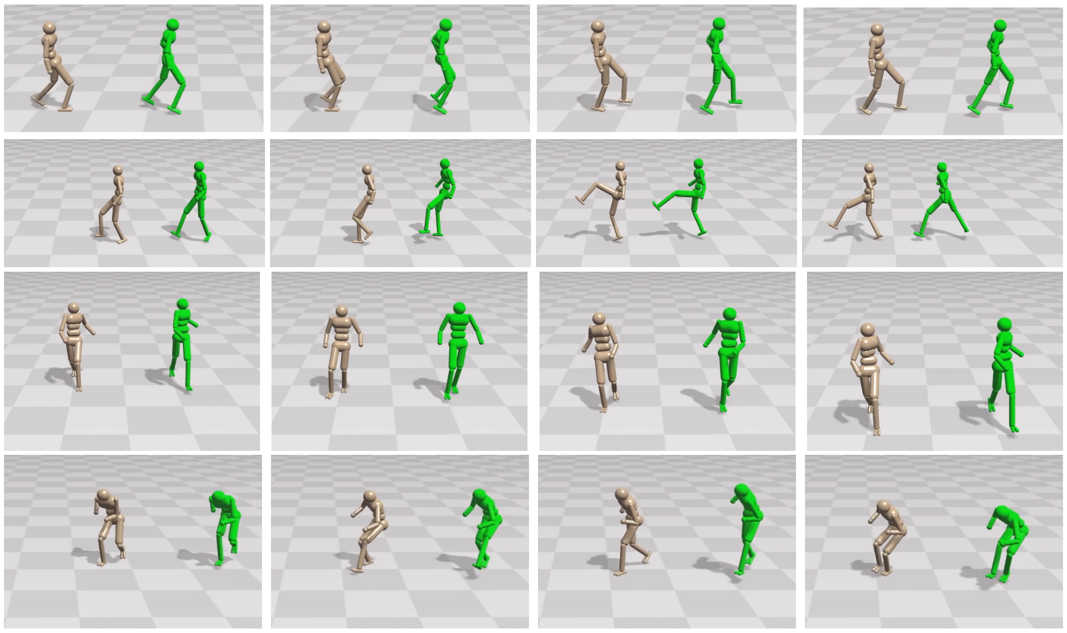

Figure 6: Normal walking, walking across a tall obstacle, walking while swinging arms, crouch walking mocap clips imitated by our agent.

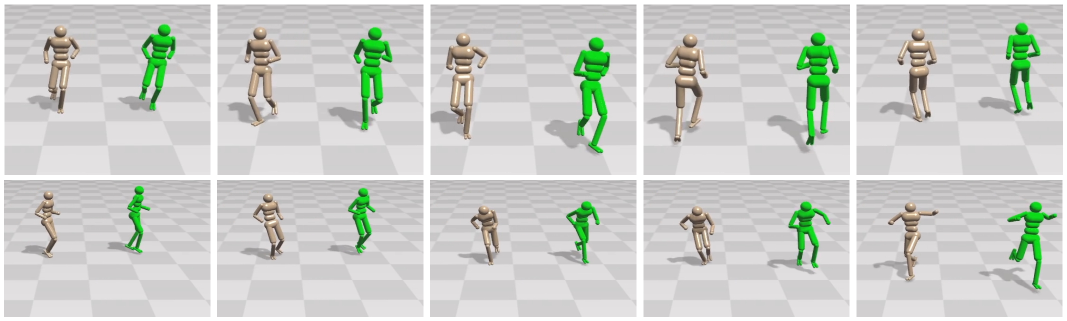

Figure 7: Unseen jogging and jumping mocap clip being imitated by our agent. The mocap clips are not used during training, demonstrating that our method can generalize to unseen clips to a certain extent.

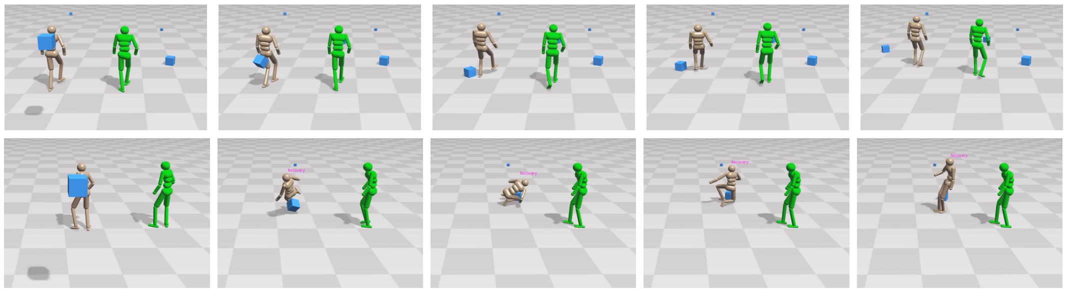

Figure 8: The character get boxes threw at. Top) The character get displaced and the tracking controller continues to control it to track the mocap clip. Bottom) The character falls down, the recovering agent controls the character to get up and the tracking agent resumes the control to continue tracking the mocap clip.

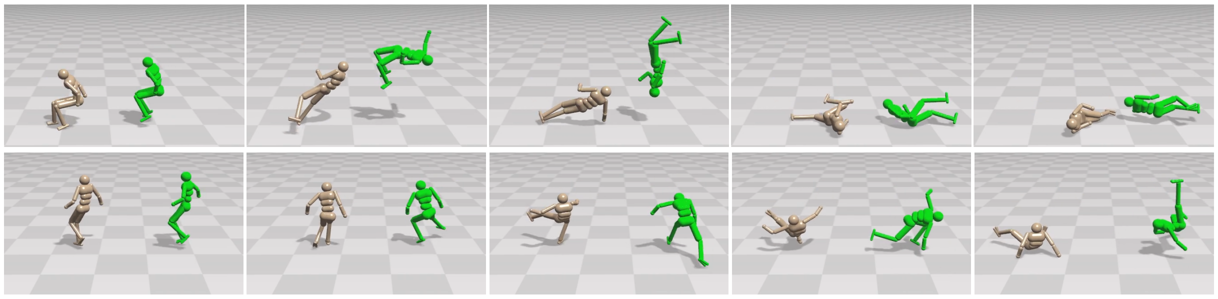
Figure 9: Backflip and break dance clips, failed to be tracked by our agent

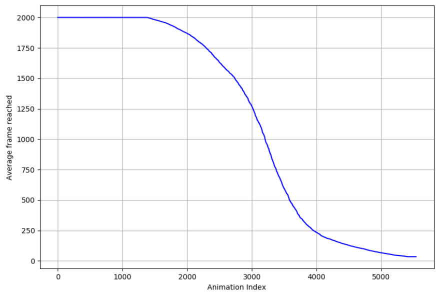

Figure 10: Frame reached before the *tracking agent* is dead, starting from the first frame, for each padded mocap clip from the database averaging over 20 trials. The animations are sorted sorted from the highest to the smallest frames reached.

is an interesting area of future work to explore curriculum based learning or some other training techniques to be able to imitate a larger portion of the motion capture database with a single network.

Our method produces jittery motion in some rare cases. This is not due to the simulation error. We noticed that the network from PPO iteration around 20,000 does not have this issue at all, but it can also imitate fewer mocap clips. We think that this is due to the agent trying to optimize for more reward by performing micro movements. This is rather undesirable and it will be an interesting future task to add a reward term that penalizes this behavior while not affecting the imitation capability.

The use of two individual networks for tracking and for recovering is not very convenient and it requires some rules to switch between them. An interesting area of future work would be to investigate on what it would take to be able to train both cases with the same network. Alternatively, a higher-level controller similar to the one used in DeepLoco [Peng et al. 2017] could be trained to switch between them seamlessly.

Our motion clips used as target for the agent mostly come from a mocap database. In addition to this, it would be interesting to use the motion graph [Kovar et al. 2002] with motion blending [Feng et al. 2012] to generate target clips. Alternatively, a real-time motion clip generator such as [Holden et al. 2017] could also be used. These methods may provide an even more diverse dataset for training the RL agent.

Training the agent to follow motion clips on non-flat terrain and object manipulation would also be an interesting area for future work. More observations and novel training methods would likely be needed to allow the agent to master such situations. Nonetheless, our *tracking agent* is able to control the character to imitate some motion clips on uneven terrain with small and moderate slope, an example of which is shown in Figure 11 and the accompanying video. For the high slope terrain, the character eventually falls down and the *recovering agent* is able to control the character to get up a few times.

REFERENCES

[n. d.]. CMU Motion Capture Database. http://mocap.cs.cmu.edu/.

[n. d.]. RoboSchool. https://github.com/openai/roboschool.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics* 19, 8 (Aug. 2013), 1405–1414.

anonymized for review. 2018. Non-Smooth Newton Methods for Deformable Multi-Body Dynamics. *In submission* (2018).

Glen Berseth, Cheng Xie, Paul Cernek, and Michiel van de Panne. 2018. Progressive Reinforcement Learning with Distillation for Multi-Skilled Motion Control. *International Conference on Learning Representations* (2018).

Joseph L. Cooper and Dana Ballard. 2012. Realtime, Physics-Based Marker Following. In *Motion in Games*, Marcelo Kallmann and Kostas Bekris (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–361.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust Task-based Control Policies for Physics-based Characters. *ACM Trans. Graph.* 28, 5, Article 170 (Dec. 2009), 9 pages.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Trans. Graph.* 29, 4, Article 130 (July 2010), 9 pages.

Taku Komura Daniel Holden, Jun Saito. [n. d.]. Motion capture database. http://theorangeduck.com/media/uploads/other_stuff/motionsynth_code.zip.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2017. OpenAI Baselines. https://github.com/openai/baselines.

Andrew Feng, Yazhou Huang, Marcelo Kallmann, and Ari Shapiro. 2012. An Analysis of Motion Blending Techniques. In *Motion in Games*, Marcelo Kallmann and Kostas Bekris (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 232–243.

T. Geijtenbeek and N. Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2492–2515.

T. Geijtenbeek, N. Pronost, and A. F. van der Stappen. 2012. Simple Data-driven Control for Simulated Bipeds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12)*. Eurographics Association, Goslar Germany, Germany, 211–219.

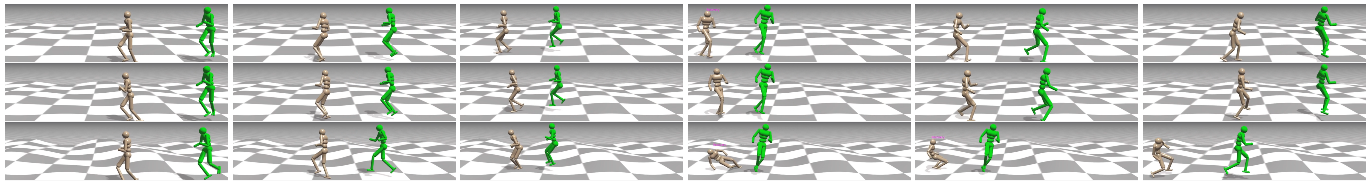

Figure 11: Walking mocap clip imitating on uneven terrain, which are not used during training. The slope of the terrain increases from top to bottom. The *tracking agent* is able to imitate this mocap clip without falling down on the low and medium slope but fall down on the high slope. The *recovering agent* is able to control the character to get up.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-based Locomotion for Bipedal Creatures. *ACM Trans. Graph.* 32, 6, Article 206 (Nov. 2013), 11 pages.

Radek Grzeszczuk and Demetri Terzopoulos. 1995. Automated Learning of Muscle-actuated Locomotion Through Control Abstraction. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 63–70.

Nikolaus Hansen. 2007. The CMA Evolution Strategy: A Comparing Review. , 75-102 pages.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. 2017. Emergence of Locomotion Behaviours in Rich Environments. *CoRR* abs/1707.02286 (2017).

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. 1995. Animating Human Athletics. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 71–78.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages.

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (July 2016), 11 pages.

Zumra Kavafoglu, Ersan Kavafoglu, Gokcen Cimen, Tolga Capin, and Hasmet Gurcay. 2018. Style-based biped walking control. *The Visual Computer* 34, 3 (01 Mar 2018), 359–375.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-Normalizing Neural Networks. *CoRR* abs/1706.02515 (2017). arXiv:1706.02515

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Trans. Graph.* 21, 3 (July 2002), 473–482.

Vikash Kumar. [n. d.]. Humanoid 1.31. http://www.mujoco.org/forum/index.php?resources/humanoid.5/.

Taesoo Kwon and Jessica Hodgins. 2010. Control Systems for Human Running Using an Inverted Pendulum Model and a Reference Motion Capture Sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 129–138.

Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. 1996. Limit Cycle Control and Its Application to the Animation of Balancing and Walking. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 155–162.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 8 pages.

Sergey Levine and Pieter Abbeel. 2014. Learning Neural Network Policies with Guided Policy Search Under Unknown Dynamics. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 1071–1079.

Sergey Levine and Vladlen Koltun. 2014. Learning Complex Neural Network Policies with Trajectory Optimization. In *ICML '14: Proceedings of the 31st International Conference on Machine Learning*.

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). arXiv:1509.02971

Libin Liu, Michiel Van De Panne, and Kangkang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Trans. Graph.* 35, 3, Article 29 (May 2016), 14 pages.

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving Sampling-based Motion Control. *Comput. Graph. Forum* 34, 2 (May 2015), 415–423.

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

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. 2017. Learning human behaviors from motion capture by adversarial imitation. *CoRR* abs/1707.02201 (2017).

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel Todorov. 2015. Interactive Control of Diverse Complex Characters with Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 3132–3140.

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

Igor Mordatch, Jack M. Wang, Emanuel Todorov, and Vladlen Koltun. 2013. Animating Human Lower Limbs Using Contact-invariant Optimization. *ACM Trans. Graph.* 32, 6, Article 203 (Nov. 2013), 8 pages.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deep-Mimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)* 37, 4 (2018).

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.* 34, 4, Article 80 (July 2015), 11 pages.

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, 4 (2016).

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 (Proc. SIGGRAPH 2017)* 36, 4 (2017).

Xue Bin Peng and Michiel van de Panne. 2017. Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17)*. ACM, New York, NY, USA, Article 12, 13 pages.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015a. Trust Region Policy Optimization. *CoRR* abs/1502.05477 (2015). arXiv:1502.05477

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347

Karl Sims. 1994. Evolving Virtual Creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. ACM, New York, NY, USA, 15–22.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating Biped Behaviors from Human Motion Data. *ACM Trans. Graph.* 26, 3, Article 107 (July 2007).

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011. Articulated Swimming Creatures. *ACM Trans. Graph.* 30, 4, Article 58 (July 2011), 12 pages.

H. van Hasselt and M. A. Wiering. 2007. Reinforcement Learning in Continuous Action Spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. 272–279.

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

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2009. Optimizing Walking Controllers. In *ACM SIGGRAPH Asia 2009 Papers (SIGGRAPH Asia '09)*. ACM, New York, NY, USA, Article 168, 8 pages.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2010. Optimizing Walking Controllers for Uncertain Inputs and Environments. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 73, 8 pages.

Y. Wang, Z. Wang, G. Bao, and B. Xu. 2014. Optimization control for biped motion trajectory. In *2014 International Conference on Audio, Language and Image Processing*. 780–785.

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