SuperTrack: Motion Tracking for Physically Simulated Characters using Supervised Learning

LEVI FUSSELL, University of Edinburgh, UK
KEVIN BERGAMIN, Ubisoft La Forge, Ubisoft, Canada
DANIEL HOLDEN, Ubisoft La Forge, Ubisoft, Canada

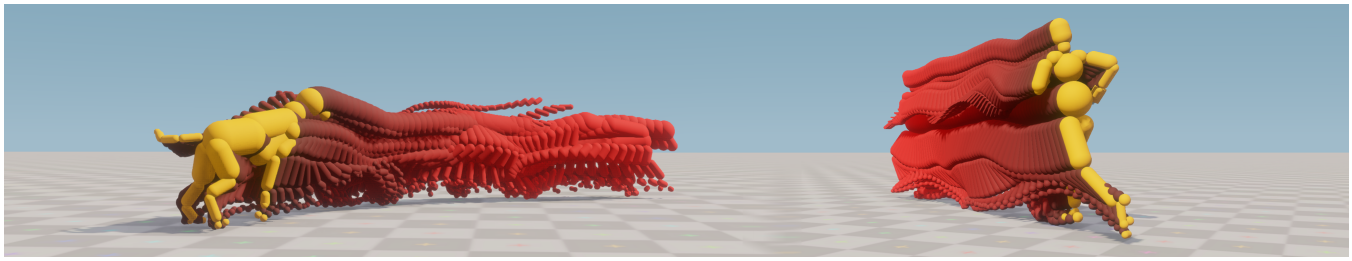

Fig. 1. Visualization of the predictions of the world model, which is used by our method as an approximate differentiable physics simulator.

In this paper we show how the task of motion tracking for physically simulated characters can be solved using supervised learning and optimizing a policy directly via back-propagation. To achieve this we make use of a world model trained to approximate a specific subset of the environment's transition function, effectively acting as a differentiable physics simulator through which the policy can be optimized to minimize the tracking error. Compared to popular model-free methods of physically simulated character control which primarily make use of Proximal Policy Optimization (PPO) we find direct optimization of the policy via our approach consistently achieves a higher quality of control in a shorter training time, with a reduced sensitivity to the rate of experience gathering, dataset size, and distribution.

CCS Concepts: • **Computing methodologies** → **Animation**.

Additional Key Words and Phrases: Motion Tracking, Motion Imitation, Imitation Learning, Reinforcement Learning, Character Animation, Motion Capture

ACM Reference Format:
Levi Fussell, Kevin Bergamin, and Daniel Holden. 2021. SuperTrack: Motion Tracking for Physically Simulated Characters using Supervised Learning. *ACM Trans. Graph.* 40, 6, Article 1 (December 2021), 13 pages. https://doi.org/10.1145/3478513.3480527

Authors' addresses: Levi Fussell, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, UK, levi.fussell@ed.ac.uk; Kevin Bergamin, Ubisoft La Forge, Ubisoft, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, kevin.bergamin@ubisoft.com; Daniel Holden, Ubisoft La Forge, Ubisoft, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, daniel.holden@ubisoft.com.

0730-0301/2021/12-ART1 $15.00
https://doi.org/10.1145/3478513.3480527

1 INTRODUCTION

In recent years Reinforcement Learning (RL) has been shown to consistently achieve state-of-the-art results on a large number of physically simulated character control tasks - and the ability to track high quality motion capture data with a physically simulated character is now well within the reach of both animation researchers and practitioners. Many of these works make use of Proximal Policy Optimization (PPO), a popular, on-policy Reinforcement Learning algorithm that has been shown to work effectively on a vast range of different problems and applications. However, like many Reinforcement Learning algorithms, PPO is notoriously sample inefficient, sensitive to hyper-parameter settings, class or data imbalance, gym design, reward and feature design, and even the random seed [Henderson et al. 2017]. This makes wide adoption of such methods in areas like video games difficult, as the unpredictability of the algorithm's success can make it a risky choice in a production setting.

In this work we show how the most common task in physically simulated character control - motion tracking - can instead be formulated using only supervised learning such that a control policy can be optimized for directly using back-propagation of a tracking loss. This can be interpreted in two ways: as a form of model-based learning, but also as a supervised learning problem where two models are learned simultaneously - one to predict the dynamics of the world, and another which attempts to produce the optimal actions which minimize the tracking losses. We find our approach achieves a better quality of animation within a shorter training time than PPO, with a reduced sensitivity to the rate of experience gathering and data imbalance, which we show on a large variety of challenging tracking tasks.

Our method relies on four key observations: first, that the reward or loss function for the motion tracking task is generally a simple, differentiable function of the kinematic and simulated character states. Second, that the transition function of these two individual character states is partially separable; generally the kinematic character's behaviour does not change because of the simulated

character's behaviour. Thirdly, that the transition function of the simulated character is entirely the result of a deterministic physics simulation, and as such can be approximated effectively by a basic feed-forward neural network. And finally, that by approximating the physics simulation with a neural network we produce an approximate differentiable physics simulator through which the policy can be optimized for directly via back-propagation of the tracking losses.

The core of our method consists of two neural networks: the *world model*, which is trained to predict the next simulated character state given the previous simulated character state and the provided PD targets, and the *policy* which is trained to produce PD offsets given the simulated and kinematic character states, with the objective that the resulting simulated character, when passed through the world model, should match the target kinematic character state. Although in theory these networks can both be trained offline given a large enough dataset, due to the vast data coverage required we set up an interactive training environment resembling an RL gym that writes data into a large cyclic buffer, training each network in tandem on random samples from this buffer.

To evaluate our method we train it on a number of different challenging tracking tasks and compare its performance to PPO in terms of motion quality, training time, and other design decisions.

2 RELATED WORK

In this section we give a brief overview of related research including model-based methods, learned world models, model-free methods, and specific works on motion tracking.

2.1 Model-based Learning

Model-based approaches utilize an underlying model of the physical system to achieve motion control. The model allows optimization to be used to solve control problems, either by directly exploiting the structure of the model itself [Brown et al. 2013; Coros et al. 2010; da Silva et al. 2008; Eom et al. 2019; Hong et al. 2019; Jain and Liu 2011; Lee et al. 2010; Macchietto et al. 2009; Muico et al. 2009; Yin et al. 2007], or by using black-box, gradient-free optimization methods [Hämäläinen et al. 2014, 2015; Lee et al. 2014; Liu et al. 2015, 2010; Naderi et al. 2017; Sok et al. 2007]. Given the model, an optimization can be performed either offline or online to find the actions which achieve the desired goals for a specific state. While gradient-free optimization can be applied more generally than optimizations designed for limited, specific domains, it often struggles when the dimensionality of the action space grows.

Differentiable models allow gradient-based optimization to be leveraged in a way which scales far better as the number of control parameters increases [Tassa et al. 2012; Todorov et al. 2012], but is hard to formulate beyond specific tasks, and even with gradients, it is not always clear how to use these models to optimize for *Policies* that achieve desired *behaviours* rather than specific actions limited to individual states [Ding et al. 2015; Liu et al. 2016]. Model-based reinforcement learning is one way to do this, and provides a framework for learning a policy from just a reward signal by optimizing for the control policy that takes actions which maximize the expected discounted return [Janner et al. 2019]. However, model-based reinforcement learning is often just as difficult to apply in the general case. Sometimes models can be designed from first principles using domain expertise [Tassa et al. 2012], or a hybrid approach can be used where an approximate parameterized model is fit to limited data (often called "system identification" [Ljung 1999]). Yet, usually as system complexity grows manual model design becomes impractical.

2.2 Learned World Models

In an attempt to obtain a differentiable model of a complex system, one approach (and indeed the one proposed in this paper) is to train a generic differentiable model such as a neural network on observations of the system in a supervised way, providing a differentiable approximation which can be used to perform gradient-based optimization of control policies [Nagabandi et al. 2017; Schmidhuber 1990]. These *World Models* [Ha and Schmidhuber 2018], can take many forms, but are often trained to approximate the *transition function* of the environment - to predict the next state given the previous state and the action taken [Chiappa et al. 2017; Dosovitskiy and Koltun 2016; Sutton 1991]. This approach, however, comes with its own challenges as many of the elements of the environment may not be included in the observations, may be stochastic in nature, or may change independently of the agent's actions. In the general case this makes learning the transition function difficult as it must be a probabilistic function [Deisenroth and Rasmussen 2011; Depeweg et al. 2017; Higuera et al. 2018], trained on enough data to break spurious correlations between actions and observations [Wahlström et al. 2015], and handle well the prediction of sparse yet important events [Oh et al. 2015]. In this work we sidestep the majority of these issues by limiting our world model to just the subset of the transition function which is affected by the agent's actions: the result of the deterministic physics simulation.

Our method can therefore be categorized as a model-based learning method that uses a learned world model. As such, Grzeszczuk et al. [1998] is perhaps the most similar previous work to our approach, as they also train a learned world model to predict the dynamics of an expensive physics model. This world model is then used to generate physically-based animation clips using gradient-based trajectory optimization. Our work could be viewed as adding an additional step that attempts to memorize the results of this optimization in the form of the policy network function.

2.3 Model-free Learning

As with the training of world models, it is often easier to make measurements of a complicated system's dynamics than it is to design an accurate, flexible model which can be exploited efficiently to build a policy that can achieve the desired behaviors. Model-free reinforcement learning exploits this fact and uses measurements of the response of various states of the environment to various actions to optimize a policy without any knowledge of the underlying dynamics. This makes it attractive for the purposes of solving physical control problems where effects like friction, air resistance, actuator dynamics, contact response, etc. are notoriously difficult to accurately model [Hwangbo et al. 2019]. However, since many model-free methods are highly sample-inefficient, they are best

suited to simulated environments where samples are easy and quick to obtain. Physics engines developed for games [Coumans 2015; Havok 2021; PhysX 2021] are a good example of this, and are often treated as black-box simulators for this purpose.

One of the most popular model-free algorithms currently used for control tasks is Proximal Policy Optimization [Schulman et al. 2017] (PPO), which has been applied in a variety of state-of-the-art control-related research on human characters [Bergamin et al. 2019; Clegg et al. 2018; Hu et al. 2020; Lee et al. 2019; Liu and Hodgins 2018; Luo et al. 2020; Merel et al. 2019a,b; Park et al. 2019; Peng et al. 2018a, 2021; Won et al. 2020; Won and Lee 2019; Xie et al. 2020; Yu et al. 2018], and has proven itself as one of the most popular on-policy learning algorithms available when sample inefficiency is of no great concern.

2.4 Motion Tracking

Many works on motion tracking have focused on the emergence of locomotion-like behaviour from simple objectives or carefully designed heuristics [Coros et al. 2010; Di Carlo et al. 2018; Geijtenbeek and Pronost 2012; Heess et al. 2017; Naderi et al. 2017; Raibert and Hodgins 1991; Sok et al. 2007; Yin et al. 2007]. However, a desire for more realistic animation has brought a focus onto the use of motion capture data [Hong et al. 2019; Liu and Hodgins 2018, 2017; Liu et al. 2016, 2015, 2010], or even video [Peng et al. 2018b], as a tracking reference. The most recent motion tracking works primarily use PPO, and provide a "tracking reward" for how closely the animation data is followed [Bergamin et al. 2019; Chentanez et al. 2018; Luo et al. 2020; Merel et al. 2019b; Park et al. 2019; Peng et al. 2018a, 2021; Won et al. 2020; Yu et al. 2018]. Surprisingly, there is very little work applying model-based RL for motion imitation, with most instances being applied on smaller problems [Zhou et al. 2019], or as benchmarks [Brockman et al. 2016]. To our knowledge, we are the first to apply a learned world model to the problem of motion tracking at such a large scale.

3 METHODOLOGY

In this section we describe the training environment, the representation we use for the simulated and kinematic characters, and the training algorithm we use to optimize the world model and policy.

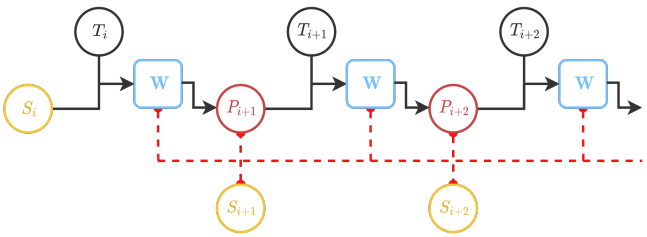

Fig. 2. Visual overview of the training procedure for the world model where red dotted lines represent the back-propagation of the losses. Here $\mathcal{W}$ is the world model, P_i is the world model predicted state, S_i is the ground truth state, T_i is the policy PD-target output.

Algorithm 1: Algorithm for training the world model $\mathcal{W}$.

Function `TrainWorldModel`(S, T, $N_{\mathcal{W}}$, $\theta_{\mathcal{W}}$, dt)**:**

```
/* Set initial predicted state                          */
P_0 ← S_0
/* Predict P over a window of N_W frames                */
for i ← 0 to N_W − 1 do
    /* Predict rigid body accelerations                 */
    p̈_i^lp, p̈_i^lr ← W([Local(P_i) T_i]^T; θ_W)
    /* Convert accelerations to world space             */
    p̈_i^p, p̈_i^r ← p_i^rootr ⊗ p̈_i^lp, p_i^rootr ⊗ p̈_i^lr
    /* Integrate rigid body accelerations               */
    P_{i+1} ← Integrate(P_i, p̈_i^p, p̈_i^r, dt)
end
/* Compute losses                                       */
L_pos ← w_pos Σ_i ‖ s_i^p − p_i^p ‖_1
L_vel ← w_vel Σ_i ‖ ṡ_i^p − ṗ_i^p ‖_1
L_rot ← w_rot Σ_i ‖ s_i^r ⊖ p_i^r ‖_1
L_ang ← w_ang Σ_i ‖ ṡ_i^r − ṗ_i^r ‖_1
/* Update network parameters                            */
θ_W ← RAdam(θ_W, ∇ Σ_* L_*)
```

end

3.1 Environment

To allow for enough data to be collected we set up a training environment that closely resembles an RL gym similar in style to that of DeepMimic [Peng et al. 2018a]. We simulate 256 different articulated characters in parallel with motors at each joint, driven by PD controllers, with the goal of tracking some corresponding kinematic animation taken from a motion capture database. Whenever some maximum episode length is exceeded, or some minimum episode length has been obtained and the character's head height has deviated from the reference by more than 25cm, reference state initialization [Peng et al. 2018a] is used to reset the simulated character to a random pose from a random animation in the kinematic animation database, with vertical correction applied to avoid ground penetrations. We use a maximum episode length of 512, a minimum episode length of 48, and provide an option to apply resets with respect to some user-provided probability distribution over animations. Once an episode terminates, the samples from the environment are added to a large cyclic data buffer (see Section 3.5 for more details). This setup generates data at a rate of ~5000 samples per second. For more details on our simulation settings please see Section 5.3.

3.2 Representation

To represent the state of a given character $\mathbf{X}$ we use the world space positions $\mathbf{x}^p \in \mathbb{R}^{B\times 3}$, velocities $\dot{\mathbf{x}}^p \in \mathbb{R}^{B\times 3}$, rotations $\mathbf{x}^r \in \mathbb{R}^{B\times 4}$, and rotational velocities $\dot{\mathbf{x}}^r \in \mathbb{R}^{B\times 3}$ of all the associated rigid bodies, $\mathbf{X} = \{\mathbf{x}^p\ \dot{\mathbf{x}}^p\ \mathbf{x}^r\ \dot{\mathbf{x}}^r\}$, where B is the number of rigid bodies. Here, rotations are represented as quaternions. We use the same representation for both the kinematic character $\mathbf{K}$ and the simulated character $\mathbf{S}$.

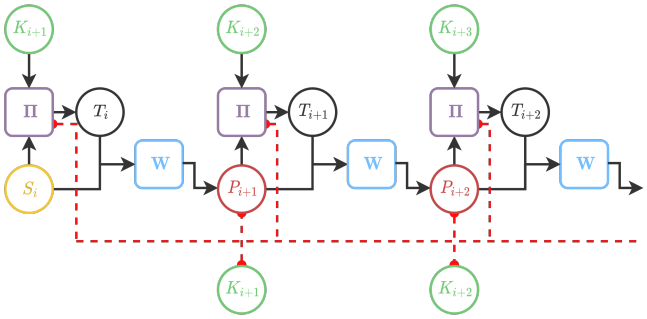

Fig. 3. Visual overview of the training procedure for the policy where red dotted lines represent the back-propagation of the losses. Here $\mathcal{W}$ is the world model, P_i is the world model predicted state, Π is the policy, T_i is the policy PD-target output, K_i is the kinematic state.

To represent the simulated character's PD targets $\mathbf{T}$ we use the target rotation $\mathbf{t} \in \mathbb{R}^{J\times 4}$ and target rotational velocity $\dot{\mathbf{t}} \in \mathbb{R}^{J\times 3}$ for each joint $\mathbf{T} = \{\mathbf{t}\ \dot{\mathbf{t}}\}$, where J is the number of joints. The PD offsets computed by the policy $\mathbf{o} \in \mathbb{R}^{J\times 3}$ are represented using vectors which are converted to rotations via the exponential map [Grassia 1998]. These are multiplied by the joint rotations of the kinematic character $\mathbf{k}^t \in \mathbb{R}^{J\times 4}$ to get the final PD targets $\mathbf{t} = \exp\left(\frac{\alpha}{2}\ \mathbf{o}\right) \otimes \mathbf{k}^t$, where α provides a way to adjust the overall scaling of the offsets. Since our method only applies offsets to the target rotations, the PD target rotational velocities $\dot{\mathbf{t}}$ are simply copied over directly from the rotational velocities of the joints of the kinematic character $\dot{\mathbf{k}}^t \in \mathbb{R}^{J\times 3}$, $\dot{\mathbf{t}} = \dot{\mathbf{k}}^t$ or set to zero in physics simulators which do not support them $\dot{\mathbf{t}} = \mathbf{0}$. See Section 5.1.3 for more information.

3.3 Network Inputs

The global space coordinates described in the previous section are inappropriate to provide to a network directly as they are not translation or rotation invariant, and the raw quaternions have issues of double-cover [Pavllo et al. 2019]. Instead, a number of pre-processing steps are required to prepare the network inputs: first, we convert the rigid body coordinates into the character space by multiplying by the inverse of the root rigid body transform (typically the hip transform). Next we convert quaternions into the two-axis rotation matrix format commonly used in neural network based methods [Zhang et al. 2018]. Then, we append to the representation the heights of all the rigid bodies from the ground as well as the up direction (in our case +Z), local to the root rigid body. This gives the network the direction of gravity and the distance of each limb to the floor, which can be used to infer contact information:

$$\text{Local}(\mathbf{X}) = \{\mathbf{x}^{lp}\ \dot{\mathbf{x}}^{lp}\ \mathbf{x}^{lr}\ \dot{\mathbf{x}}^{lr}\ \mathbf{x}^{h}\ \mathbf{x}^{up}\}, \tag{1}$$

$$\mathbf{x}^{lp} \in \mathbb{R}^{B\times 3} = \text{Inv}(\mathbf{x}^{rootr}) \otimes (\mathbf{x}^{p} - \mathbf{x}^{rootp}), \tag{2}$$

$$\dot{\mathbf{x}}^{lp} \in \mathbb{R}^{B\times 3} = \text{Inv}(\mathbf{x}^{rootr}) \otimes \dot{\mathbf{x}}^{p}, \tag{3}$$

$$\mathbf{x}^{lr} \in \mathbb{R}^{B\times 6} = \text{TwoAxis}(\text{Inv}(\mathbf{x}^{rootr}) \otimes \mathbf{x}^{r}), \tag{4}$$

$$\dot{\mathbf{x}}^{lr} \in \mathbb{R}^{B\times 3} = \text{Inv}(\mathbf{x}^{rootr}) \otimes \dot{\mathbf{x}}^{r}, \tag{5}$$

$$\mathbf{x}^{h} \in \mathbb{R}^{B} = \text{Height}(\mathbf{x}^{p}), \tag{6}$$

$$\mathbf{x}^{up} \in \mathbb{R}^{3} = \text{Inv}(\mathbf{x}^{rootr}) \otimes [0\ 0\ 1]^T. \tag{7}$$

Here "TwoAxis" converts from quaternion to rotation matrix and extracts the first two columns, "Height" computes the heights of the rigid bodies relative to the ground plane or height-map, and $\otimes$ represents quaternion-quaternion multiplication or quaternion-vector product when multiplying by a vector.

Finally, before being provided as input to the network, we normalize each quantity by its mean and standard deviation which we compute from a database of kinematic data offline before training.

Algorithm 2: Algorithm for training the policy Π.

Function TrainPolicy($\mathbf{S}_0$, $\mathbf{K}$, $\mathcal{W}$, N_Π, θ_Π, σ, α, dt):
- /* Set initial predicted state */
- $\mathbf{P}_0 \leftarrow \mathbf{S}_0$
- /* Predict $\mathbf{P}$ over a window of N_Π frames */
- **for** $i \leftarrow 0$ **to** $N_\Pi - 1$ **do**
  - /* Predict PD offsets */
  - $\mathbf{o}_i \leftarrow \Pi([\text{Local}(\mathbf{P}_i), \text{Local}(\mathbf{K}_{i+1})]^T; \theta_\Pi)$
  - /* Add noise to offsets */
  - $\hat{\mathbf{o}}_i \leftarrow \mathbf{o}_i + \sigma\, \mathcal{N}(\mathbf{0}, \mathbf{1})$
  - /* Compute PD targets */
  - $\mathbf{T}_i \leftarrow \{\exp\left(\frac{\alpha}{2}\ \hat{\mathbf{o}}_i\right) \otimes \mathbf{k}^t_{i+1},\ \dot{\mathbf{k}}^t_{i+1}\}$
  - /* Pass through world model */
  - $\ddot{\mathbf{p}}^{lp}_i,\ \ddot{\mathbf{p}}^{lr}_i \leftarrow \mathcal{W}([\text{Local}(\mathbf{P}_i)\ \mathbf{T}_i]^T; \theta_\mathcal{W})$
  - /* Convert accelerations to world space */
  - $\ddot{\mathbf{p}}^{p}_i,\ \ddot{\mathbf{p}}^{r}_i \leftarrow \mathbf{p}^{rootr}_i \otimes \ddot{\mathbf{p}}^{lp}_i,\ \mathbf{p}^{rootr}_i \otimes \ddot{\mathbf{p}}^{lr}_i$
  - /* Integrate rigid body accelerations */
  - $\mathbf{P}_{i+1} \leftarrow \text{Integrate}(\mathbf{P}_i,\ \ddot{\mathbf{p}}^{p}_i,\ \ddot{\mathbf{p}}^{r}_i,\ dt)$
- **end**
- /* Compute Local Spaces */
- $\mathbf{P}^l,\ \mathbf{K}^l \leftarrow \text{Local}(\mathbf{P}),\ \text{Local}(\mathbf{K})$
- /* Compute losses in Local Space */
- $\mathcal{L}_{lpos} \leftarrow w_{lpos}\ \sum_i \left\| \mathbf{p}^{lp}_i - \mathbf{k}^{lp}_i \right\|_1$
- $\mathcal{L}_{lvel} \leftarrow w_{lvel}\ \sum_i \left\| \dot{\mathbf{p}}^{lp}_i - \dot{\mathbf{k}}^{lp}_i \right\|_1$
- $\mathcal{L}_{lrot} \leftarrow w_{lrot}\ \sum_i \left\| \mathbf{p}^{lr}_i - \mathbf{k}^{lr}_i \right\|_1$
- $\mathcal{L}_{lang} \leftarrow w_{lang}\ \sum_i \left\| \dot{\mathbf{p}}^{lr}_i - \dot{\mathbf{k}}^{lr}_i \right\|_1$
- $\mathcal{L}_{hei} \leftarrow w_{hei}\ \sum_i \left\| \mathbf{p}^{h}_i - \mathbf{k}^{h}_i \right\|_1$
- $\mathcal{L}_{up} \leftarrow w_{up}\ \sum_i \left\| \mathbf{p}^{up}_i - \mathbf{k}^{up}_i \right\|_1$
- $\mathcal{L}_{reg} \leftarrow w_{lreg}\ \sum_i \| \mathbf{o}_i \|_2^2$
- $\mathcal{L}_{sreg} \leftarrow w_{sreg}\ \sum_i \| \mathbf{o}_i \|_1$
- /* Update network parameters */
- $\theta_\Pi \leftarrow \text{RAdam}(\theta_\Pi, \nabla \sum_* \mathcal{L}_*)$

end

3.4 World Model

To train the world model $\mathcal{W}$ first we sample a window of $N_{\mathcal{W}} = 8$ frames from the data buffer. We then extract the simulated states $\mathbf{S}$, and the PD targets $\mathbf{T}$ which were used at those simulation states. Starting with the first frame in the buffer $\mathbf{S}_0$, the state and PD targets are input into the network $\mathcal{W}$, and the rigid body positional, and rotational accelerations local to the root rigid body are computed as output $\ddot{\mathbf{x}}^{lp} \in \mathbb{R}^{B\times 3}$, $\ddot{\mathbf{x}}^{lr} \in \mathbb{R}^{B\times 3}$. These are then converted into the world space $\ddot{\mathbf{x}}^{p}$, $\ddot{\mathbf{x}}^{r}$ by multiplying by the root rigid body rotation, and integrated with the current state to find the next state:

$$\text{Integrate}(\mathbf{X},\ \ddot{\mathbf{x}}^{p},\ \ddot{\mathbf{x}}^{r},\ dt) = \{\mathbf{y}^{p}\ \dot{\mathbf{y}}^{p}\ \mathbf{y}^{r}\ \dot{\mathbf{y}}^{r}\}, \tag{8}$$

$$\dot{\mathbf{y}}^{p} = dt\ \ddot{\mathbf{x}}^{p} + \dot{\mathbf{x}}^{p}, \tag{9}$$

$$\dot{\mathbf{y}}^{r} = dt\ \ddot{\mathbf{x}}^{r} + \dot{\mathbf{x}}^{r}, \tag{10}$$

$$\mathbf{y}^{p} = dt\ \dot{\mathbf{y}}^{p} + \mathbf{x}^{p}, \tag{11}$$

$$\mathbf{y}^{r} = \exp\left(\tfrac{dt}{2}\ \dot{\mathbf{y}}^{r}\right) \otimes \mathbf{x}^{r}. \tag{12}$$

This next state is then fed into the network again and the process continues until a whole window of predicted states has been computed. The difference between this prediction $\mathbf{P}$ and the ground truth $\mathbf{S}$ is then computed, and network weights updated to minimize this loss. For more details please see Algorithm 1 and Fig 2. In the loss function, weights w_{pos}, w_{vel}, w_{rot}, w_{ang}, are tweaked to give roughly equal contribution from all losses at the beginning of training and $\ominus$ represents quaternion difference - i.e. taking the log of the result of the left-hand-side quaternion multiplied by the inverse of the right-hand-side quaternion. This window-based training scheme can be interpreted as teacher-forcing [Williams and Zipser 1989] every $N_{\mathcal{W}}$ frames. While this algorithm is presented for a single sample, training is performed on mini-batches. For a visualization of trained world model predictions please see Fig 1.

3.5 Policy

To train the policy $\mathbf{\Pi}$ we first sample a window of $N_{\Pi} = 32$ frames from the data buffer, extracting the initial simulated state $\mathbf{S}_0$ as well as target kinematic states $\mathbf{K}$. Then, the simulated state and initial target kinematic state are fed into the policy to get the PD offsets $\mathbf{o}$. Gaussian noise scaled by $\sigma = 0.1$ is added to the offsets to encourage state and action trajectory exploration, and the resulting offsets $\hat{\mathbf{o}}$ are scaled by an overall scaling factor α before being converted to quaternions and multiplied by the kinematic character joint rotations $\mathbf{k}^t$. These are then paired with the kinematic character joint rotational velocities $\dot{\mathbf{k}}^t$ to produce the final PD targets $\mathbf{T}$. These PD targets are then fed into the world model, along with the predicted simulated state, to produce the next simulated state. This process is then repeated until a full window of predicted states $\mathbf{P}$ have been produced. The difference between this prediction and the target kinematic states $\mathbf{K}$ is then computed in the local space, and the losses used to update the weights of the policy. For more details, please see Algorithm 2 and Fig 3. In the loss function, weights w_{lpos}, w_{lvel}, w_{lrot}, w_{lang}, w_{hei}, w_{up} are set to give roughly equal contribution from all losses at the beginning of training while regularization weights w_{lreg} and w_{sreg} are given a smaller contribution by two orders of magnitude and are used to penalize large PD offsets. While this algorithm is presented for a single sample, training is performed on mini-batches. This algorithm also matches the procedure performed to gather training data in the training gym, but with the real physics simulator used to obtain the next simulated state rather than the world model.

3.6 Training

Our full training procedure is as follows: each iteration, first we pull any new data from the environment into a large cyclic data buffer of ~150,000 samples, which contains the simulated character $\mathbf{S}$, kinematic character $\mathbf{K}$, and PD targets used $\mathbf{T}$. Next, we transfer the current version of the policy $\mathbf{\Pi}$ to the environment. Finally, we train the world model $\mathcal{W}$ for a single step using Algorithm 1, and the policy $\mathbf{\Pi}$ for a single step using Algorithm 2. This whole process is then repeated. We start to see basic balancing behaviour after about ~10,000 iterations (which requires about two hours training on a mid-tier graphics card such as a Nvidia GTX 1070), but we found full training required ~100,000 to ~200,000 iterations, and took closer to ~20 to ~40 hours depending on the exact hardware, training setup, and desired tracking performance. For full details of the hyper-parameters used please see Table 1 and for more details on training times please see Table 2.

Table 1. Hyper-parameter settings used by our method.

World Model	Hidden Layers	5
	Hidden Units	1024
	Activation	ELU
	Batchsize	2048
	Learning Rate	0.001
	Window ($N_{\mathcal{W}}$)	8
Policy	Hidden Layers	5
	Hidden Units	1024
	Activation	ELU
	Batchsize	1024
	Learning Rate	0.0001
	Window (N_{Π})	32
Other	Noise Scale (σ)	0.1
	Offset Scale (α)	120

4 RESULTS

In this section we show the results of our method on several different animation databases as well as on interactive control. All results are shown using the PhysX [PhysX 2021] physics simulator unless otherwise mentioned. For more details please see supplementary video and for full details on the exact specifications of all the different databases used please see Table 2. To measure the success of our method, we compute the *Survival Rate*, which is defined as the percentage of episodes that last longer than a given time without the character reaching a failed state (shown in Fig 4). A failed state is as described in Section 3. Here, if the character reaches the end of the animation without failing we reset it to a new random frame without adjusting the tracking timer and continue the episode. We observe that the *Survival Rate* tends to reflect the overall size, physical difficultly, and diversity of the database.

Table 2. Details of the various databases tested.

Database	**#Frames**	**#Bodies** (B)	**#Joints** (J)	**FPS** (Hz)	**Training** (hr)
LaFAN	1,442,572	20	19	60	40
Styles	378,200	20	19	60	20
Dance	20,844	20	19	60	20
DReCon	38,332	20	19	60	20
Controller	71,322	20	19	60	20
Dog	81,240	26	25	60	20
Rough Terrain	1,544,920	20	19	60	50

4.1 Motion Tracking

In Fig 5 we show the results of our method trained on the LaFAN database [Harvey et al. 2020], a large, challenging, varied database consisting of ~6.5 hours of unstructured animation including dancing, falls and get-ups, fighting, sports, crawling, rolling, hopping, stumbles, and locomotion. In this case we limit our training to just motions on the flat ground and re-target all animation to a uniform character size. However, it should be noted that there are still many motions in the database which are physically impossible to track, such as one character pulling another character up by holding hands. Unlike existing methods we achieve high quality animation tracking without requiring the database to be broken into clusters or the training of specific experts for specific types of motions [Won et al. 2020] since we share the same policy across the whole database.

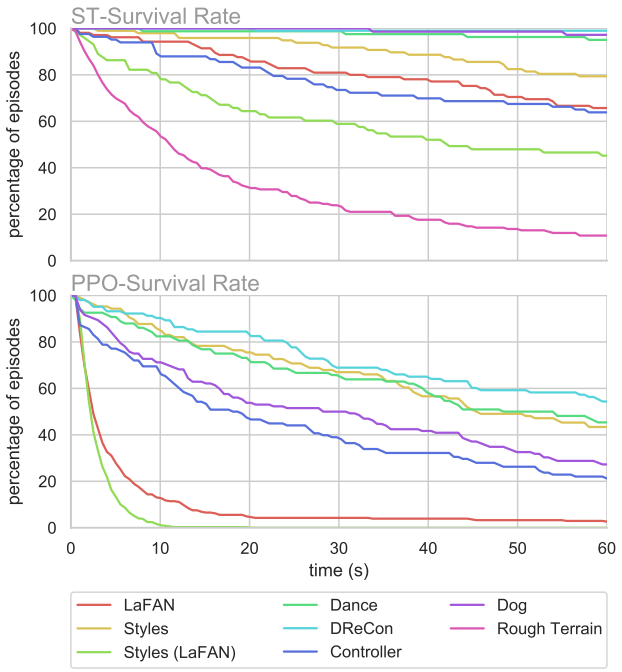

Fig. 4. Plots showing the *Survival Rate*, or percentage of episodes which last longer than a given time before failing, for various policies trained on different databases. Here *Styles (LaFAN)* represents the policy trained on the LaFAN database but tested on the *Styles* database to assess generalization ability. PPO was not tested on the Rough Terrain database.

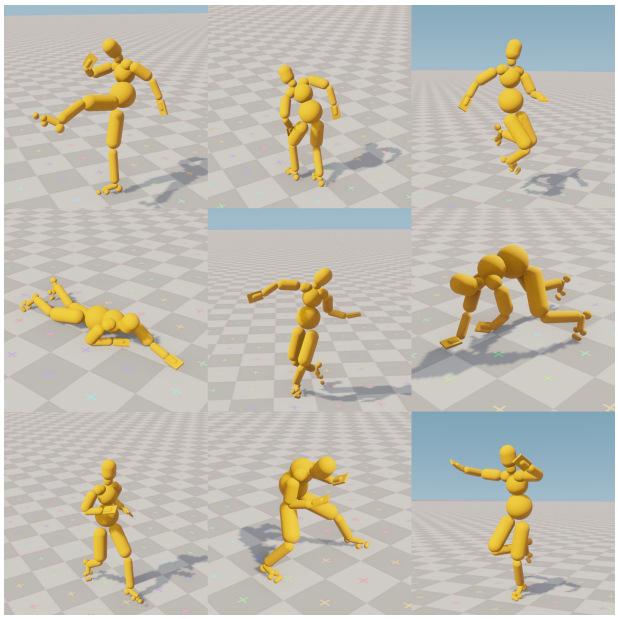

Fig. 5. Our method applied to a selection of challenging motions from the *LaFAN* database.

We also train our method on a medium sized database consisting of ~1.7 hours of locomotion in 30 different *Styles*. After training, over 80% of episodes last more than a minute.

In Fig 6 we show the results of our method trained to track a small ~6 minute database of athletic *Dance* motions. Here we achieve high quality, fluid motion even for these difficult and dynamic movements, with over 95% of episodes lasting more than a minute.

4.2 Rough Terrain

Our method and state representation can extend to tracking animations navigating *Rough Terrain*. In Fig 7 we show results of this, with a policy trained on a very large database consisting of ~7 hours of locomotion over both flat and rough terrain. This setup is particularly challenging due to the fact that the simulated character cannot be allowed to drift far from the kinematic character in the world space, or the kinematic reference will no longer be appropriate for the terrain around the simulated character's position. We therefore provide the policy with the kinematic and simulated character world space difference in root position and rotation, add additional losses

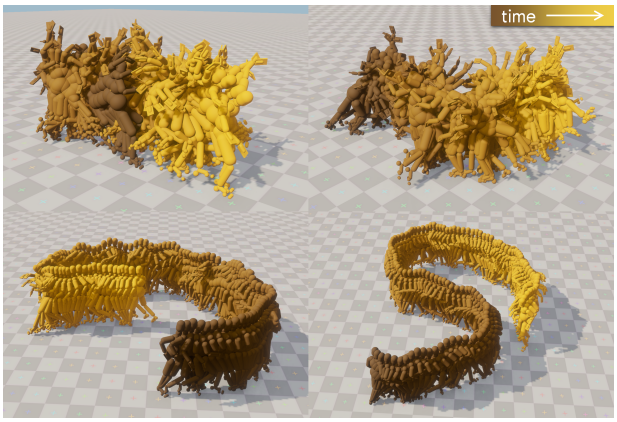

Fig. 6. Results of our method on the *Dance* database.

penalizing this difference (so that the character learns to correct any global drift), and terminate episodes as soon as the root position or rotation deviate by more than 1 meter or 90°.

4.3 Interactive Control

By pre-recording 10 to 20 minutes of user interaction with a kinematic controller, storing the resulting animation in a database, and training a policy to track it, we can build policies suitable for interactive control. In Fig 8 we show two examples of this, one responsive kinematic Motion Matching controller, and a *Dog* controller similar to the one shown in Holden et al. [2020]. We also include in our results a Motion Matching controller built from the LaFAN database similar to the one shown in DReCon [Bergamin et al. 2019].

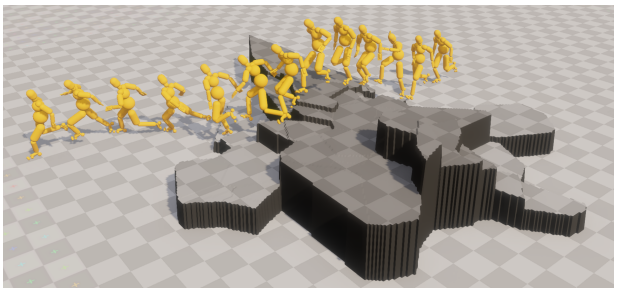

Fig. 7. Results of our method tracking animation over rough terrain.

5 EVALUATION

In this section we perform an evaluation of our method, including a detailed comparison against PPO, a study of how our method performs in a transfer learning setting, and an evaluation of the memory usage and performance. For comparing the performance of policies, we track the average episode length over the course of training. We scale this average value to be in the range $[0, 1]$ by dividing by the max episode length (in our case 512). A value of 1 therefore indicates a 100% completion rate for episodes. For all comparisons we use the same hardware for a fair evaluation (a Nvidia GTX 1070).

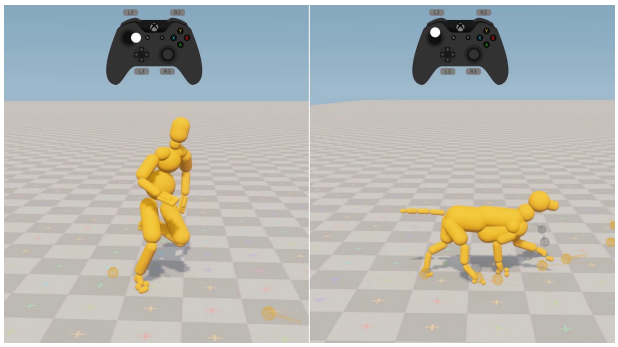

Fig. 8. Our method applied to iteractive control.

5.1 Comparison to PPO

In this section we perform a range of comparisons between our approach and PPO. These comparisons include examining variance due to changing of the random seed, behaviour when re-scaling losses or rewards, changing of the physics simulator, large data imbalance, experience gathering rate, policy synchronization rate, and robustness to perturbations. All comparisons are performed using the PhysX simulator and on the LaFAN database unless otherwise mentioned. Overall we found that while PPO performed well on smaller databases, on LaFAN it did not perform nearly as well, presumably due to the size, diversity, and range of motions contained within. See Fig 4 for more details.

Our PPO policy uses the same state and action representation as described in previous sections, but with a slightly smaller network size. Although previous works have found larger network sizes to perform both better [Wang et al. 2020] and worse [Bergamin et al. 2019], we found only a very minor trend toward increased performance from smaller networks (see Fig 10). For the policy action distribution we use a Gaussian with a fixed standard deviation scaled by σ, similar to Algorithm 2, but set to the slightly higher value of 0.2 as we found it improved performance. The gym environment, including the use of reference state initialization, is identical to the gym used for our approach. The reward structure is similar to that of Bergamin et al. [2019] but incorporates some additional terms to more closely resemble the losses used in this paper:

$$r = w_{rew} \ \exp\left(-(\mathcal{L}_{lpos} + \mathcal{L}_{lvel} + \mathcal{L}_{lrot} + \mathcal{L}_{lang} + \mathcal{L}_{lhei} + \mathcal{L}_{lup})\right)$$

where all terms here are the same as described in Algorithm 2 but with weights slightly adjusted to give a better balance of contribution (they are calculated for a single frame rather than a window of frames), and $w_{rew} = 1 \,/\, \sqrt{\frac{0.05}{1-\gamma}}$ is used to scale the overall reward. All other hyper-parameters for training PPO are outlined in Table 3.

5.1.1 Random Seed. In Fig 9 we show results of training each algorithm on three different random seeds. Overall, both PPO and our method (ST) are relatively consistent and don't show much variance across runs. This could be due to the reward definition which imposes quite strict constraints on the motion and may therefore lead to fewer local minima.

5.1.2 Loss Scale. We found both methods were relatively robust to scaling of the loss terms. In Fig 9 we show one example where we increase the scale of the w_{lvel} loss term by a factor of ×5. Again, PPO's reward definition being a multiplication of terms likely lessens the effect of any one loss component having more influence on the motion imitation than another.

5.1.3 Physics Simulator. In Fig 9 we compare results across three different simulators: PhysX [PhysX 2021], Havok [Havok 2021], and Bullet [Coumans 2015]. Since Bullet and Havok do not easily support PD target rotational velocities we set these to zero instead of using the rotational velocities of the joints of the kinematic character. We find our method's final performance is reasonably similar across simulators while PPO's final performance varies a little more, potentially due to the different rates of experience gathering. See supplementary video for a visual comparison.

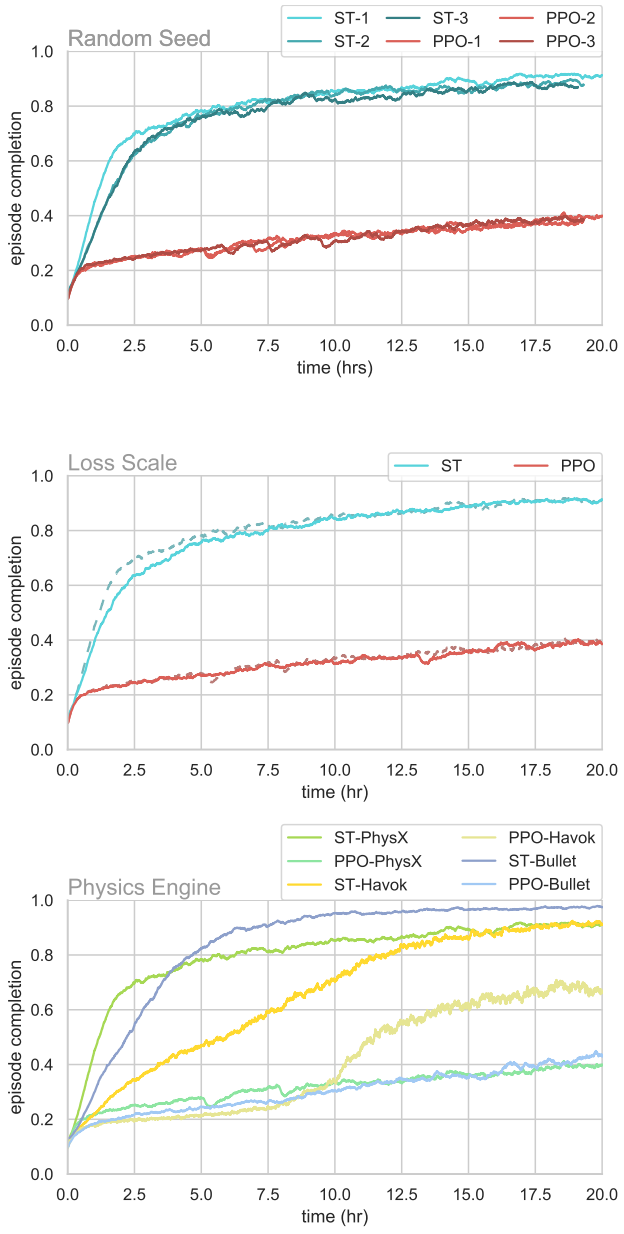

Fig. 9. Comparison of our method and PPO in terms of changing the random seed, loss scale, and simulator. Dotted lines indicate the baseline performance of the algorithms.

Table 3. Hyper-parameters for PPO.

Hidden Layers	2
Hidden Units	256
Activation	ELU
Policy Learning Rate	0.0001
Critic Learning Rate	0.001
Discount Factor (γ)	0.99
GAE-λ	0.95
Sample Size	8192
Batchsize	512
Noise Scale (σ)	0.2
Offset Scale (α)	120

5.1.4 Data Imbalance. In this experiment we take the original database used to make the Motion Matching *Controller* shown in our results, which has tags for different motion types such as *Idle*, *Walk*, *Crouch*, *Jog*, and *Run*. We then over-sample animations tagged as *Walk* by a factor of ×10. Our method is robust to changes in the tag distribution and stable run, jog, and crouch motions appear after about 1.5 hours. When changing the tag distribution for PPO, even after 10 hours of training, no motion other than walking is stable. Fig 11 shows this result and marks the point where our method had learned stable motions other than walking.

5.1.5 Experience Gathering. In this experiment we artificially reduce the rate of experience gathering in our gym environment by a factor of about ×3 by throwing away 70% of gathered episodes at random. In Fig 11 we can see that PPO trains twice as slowly while our method is largely unaffected. This is due to the off-policy and sample-efficient nature of our method compared to PPO, which we found was often bounded in performance by the rate of experience gathering in our experiments.

5.1.6 Policy Synchronization. Our method is off-policy - neither the world model nor the policy require the generated data to come from the same policy. To show this, we perform an experiment where the policy used to sample data from the gym is not the same as the most recently updated policy. Instead, we use a policy for the gym that is N training steps behind the current trained policy and we refer to N as the policy synchronization rate. As expected PPO, being an on-policy algorithm cannot not function under these conditions yet our method is largely unaffected by the synchronization interval. Fig 9 shows the results of training on policy synchronisation periods of 15 and 100 iterations.

5.1.7 Robustness. We evaluate the robustness to perturbations of policies that were trained for 20 hours with both PPO and our method. Cubes of increasing mass are thrown at the character at

a rate of once per second with a speed of 5 m/s and we measure the resulting episode completion. In Fig 13 we see that given equal training time PPO is not more robust than our method.

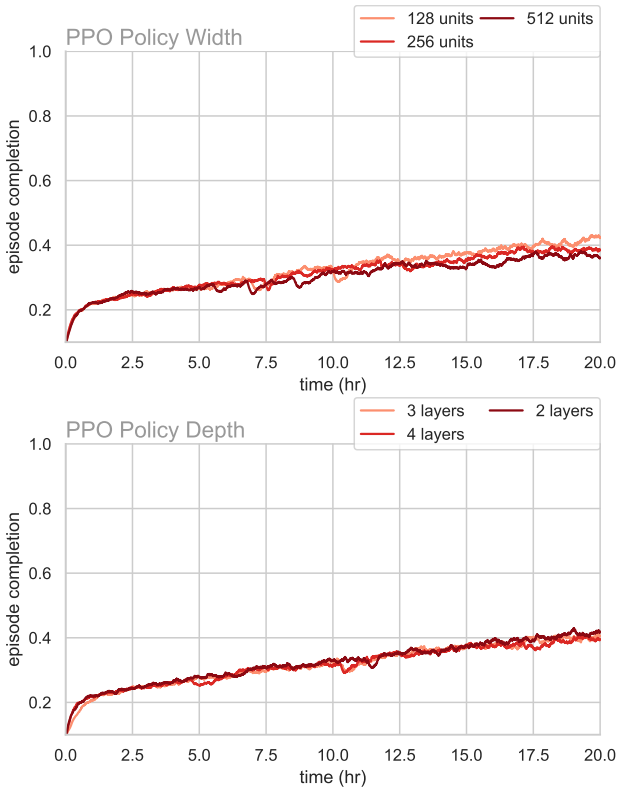

Fig. 10. Comparison of different model sizes for PPO. For policy width we use a depth of two hidden layers, for the policy depth we use 128 hidden units.

5.2 Generalization & Transfer Learning

Unlike RL methods which can often struggle to generalize and transfer to new tasks [Bengio et al. 2020], both the world model and policy in our setup can generalize to, and be fine-tuned on, new databases. To test the generalization performance we apply the more generic *LaFAN* policy to the *Styles* database (denoted "Styles (LaFAN)" in Fig 4) and find it can track well a considerable part of the database, none of which it has seen during training.

We can also fine-tune previously trained policies to new tasks. In Fig 12 we show how initializing the policy and world model of the *Styles* task with a more generic *LaFAN* policy can accelerate training. For transferring the policy, we freeze all weights except the final layer and set the policy learning rate to a very small value so that the policy does not change drastically during training. The world model initialisation and training does not change. We also test the ability to transfer just the world model and train a new policy from scratch. Here we can make use of a higher policy learning rate due to having an accurate world model early in training and the world model learning rate is reduced to prevent instability.

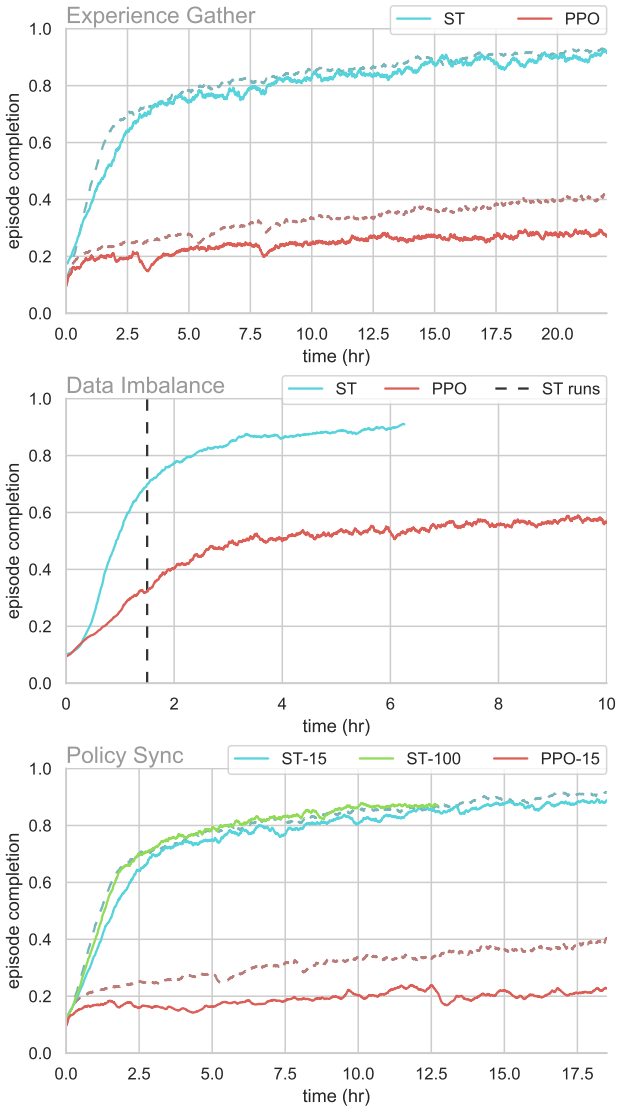

Fig. 11. Comparison of our method and PPO on experience gathering rate, data imbalance, and policy synchronization. Dotted lines indicate the baseline performance of the algorithms. For "Data Imbalance", a vertical line marks the approximate time when our method was able to performing stable running motions.

5.3 Performance

In general our method has similar performance characteristics to existing methods of physically simulated character control using PPO. Policy network evaluation takes ~1800μs, and requires ~20MB of storage for the network weights. World model performance and memory usage is similar. We run our physics simulations at 240 Hz, with 20 iterations of the constraint solver, which typically takes between 300μs and 1000μs per-frame depending on the simulator used. All performance tests were run single-threaded on an Intel Xeon E5-1650 CPU.

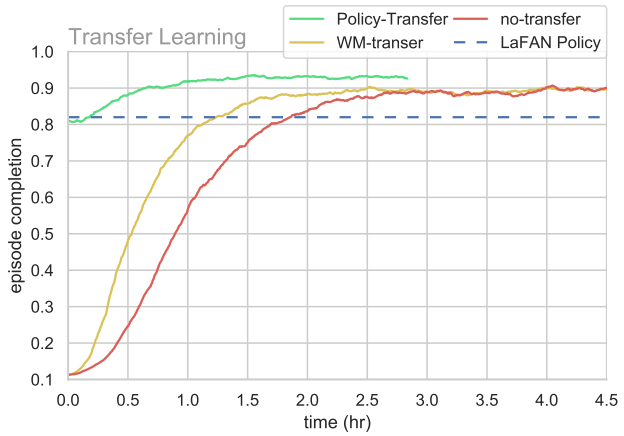

Fig. 12. Transfer learning from the *LaFAN* database to the *Styles* database. We explore both transferring the world model, and transferring the world model and the policy. The horizontal line indicates the untrained LaFAN policy performance on the *Styles* database.

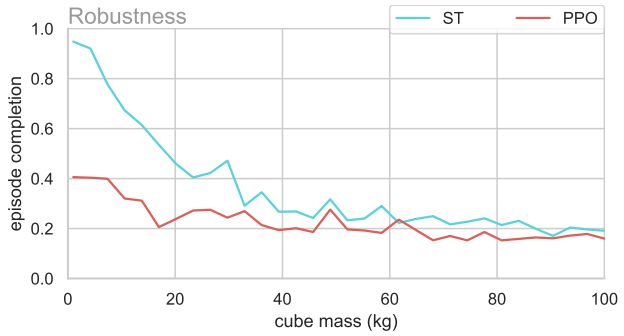

Fig. 13. Comparison of our method to PPO in terms of robustness to perturbations. We find our method produces policies more robust over a range of perturbation sizes.

6 ABLATION

In this section we present an ablation study for some of the important aspects of our method. These include the window size for both the policy and world model, the use of PD offsets rather than absolute PD targets, and the prediction of accelerations rather than velocities in the world model. As before, we try to keep the hardware consistent within experiments where it is important for the analysis. We indicate in the experiment if the hardware differs within the experiment.

6.1 Window Size

A window-based training scheme allows a model to learn longer-range predictions in a more stable way than single-step predictions. For the policy training, a larger window is needed for learning motions where long-term apprehension is required, such as taking a step in a walk cycle. However, the quality of the policy's actions over these longer-term horizons relies on the world model's long term prediction accuracy. Meanwhile, having a world model training window size that is too large can decouple the predicted states from the actions the policy took at those states and can cause the world model to ignore the actions. Fig 15 demonstrates the effect of changing the relative training window size of the world model and policy. If we set the policy window to 64 the training becomes unstable and terminates. We found that a policy window of 32 and world model window of 8 were consistently stable across all the tasks we tried.

6.2 PD Offsets

In our setup, the policy predicts PD offsets which are converted to quaternions and multiplied by the kinematic character joint rotations. Alternatively, the policy could directly predict the PD targets without using the kinematic character joint rotations as a basis, as done in some previous work [Peng et al. 2018a; Won et al. 2020]. Fig 15 shows the result of this, and confirms that predicting the offsets is more accurate and learns faster as the kinematic joint rotations provide a good starting point close to the target motion.

6.3 World Model Accelerations

The world model predicts the accelerations of the rigid bodies and this is integrated to compute the velocities and then the positions. Alternatively, the world model could directly predict the velocities. Fig 15 compares these two methods and shows the effectiveness of using the accelerations.

7 LIMITATIONS

In our method certain settings such as having the learning rate too high or low, or having window sizes $N_{\mathcal{W}}$, N_{Π} set poorly can negatively affect the training time or stability - and with an already long training time - even an adjustment that causes training to take twice as long can quickly make running tests and experiments impracticable.

Our method is also only shown on the control task of motion tracking, and while we believe it should be possible to extend it to achieve any goal which does not change due to the simulated character's behaviour and can be expressed as a differentiable function of the character states, this is not something we tested.

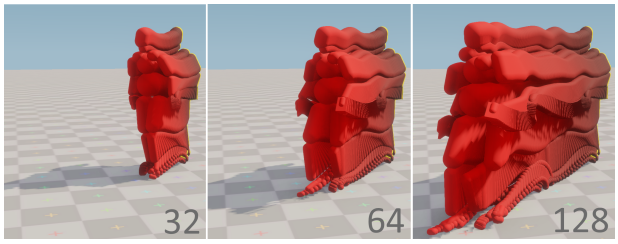

Fig. 14. Visualisation of the trained world model rollouts for 32, 64, and 128 frames. Beyond 64 frames, the world model struggles to maintain the joint constraints for accurate pose prediction.

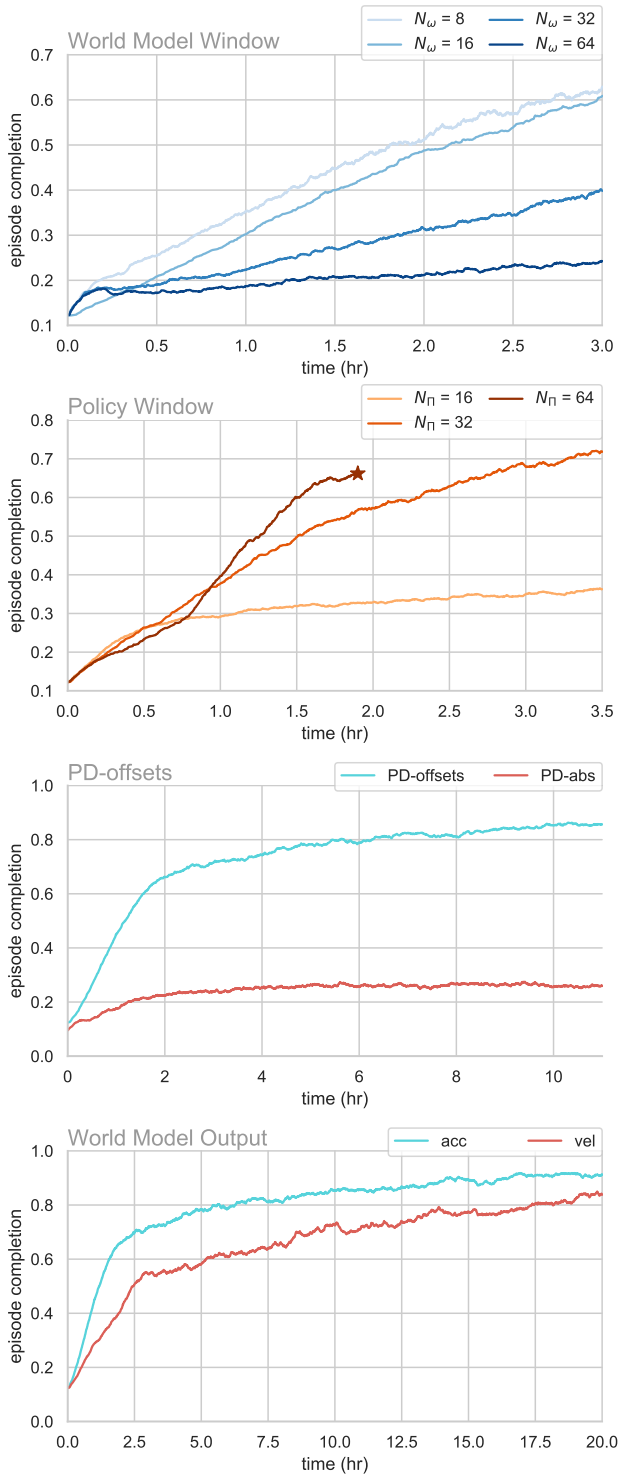

Fig. 15. Results of the ablation experiments from top to bottom: varying the window sizes for the policy & world model, predicting PD offset compared to absolute PD targets, and predicting rigid body accelerations compared to velocities using the world model.

Unlike RL based methods which have a theoretically infinite look-ahead horizon controlled by the discount factor, the look-ahead of our method is limited to the policy window size N_Π. This means high-level tasks which require apprehension at a larger time scale than the window size will be very difficult for our method to learn. Simply increasing the window size is not a good solution as it increases the instability of training due to the recurrent, auto-regressive feedback of the prediction and errors in the world model prediction will compound if the rollout is too large, as shown in Fig 14. Although we found gradient clipping to help somewhat in this respect, our method is still most likely limited to control tasks with fairly short-term, dense rewards such as motion tracking.

8 DISCUSSION AND FUTURE WORK

While PPO utilizes empirical estimates of the gradients to optimize the policy, our approach computes an analytical gradient. This has both pros and cons - on the one hand we are capable of achieving accurate tracking for all rigid bodies and their velocities since our policy always moves in the direction which best optimizes all the different loss components together. Empirically our method also seems to scale better to larger network sizes. On the other hand, our method does not exploit exploration in the same way as PPO, and it can be more difficult for it to discover novel behaviours such as stepping motions to recover from falls, as these kinds of behaviours do not typically lie on a direct downward gradient from the current policy and can require a more exploratory search method to discover them. This means artifacts such as foot sliding can appear in some motions where the short-horizon motion tracking loss is not able to plan as far ahead as PPO's longer-horizon returns.

Stable and accurate world model rollouts are necessary for our method to be successful. In order to achieve this in the high dimensional state space it's important to use a translation and rotation-invariant state representation as well as a window-based training scheme. Similarly, by limiting the world model to learning only the subspace of possible dynamics close to the kinematic character's movements we can significantly simplify the learning task. The stability of long rollouts could potentially be improved in the future by additionally learning a compressed, latent, encoding of the character state and performing the physical integration in this space.

We found that during inference it's possible to use our world model as an approximate replacement of the real physics simulation it is trained on. However, in our case this is often more expensive than the original simulation due to the relatively large network size.

In our setup we only predict the PD target rotation offsets, but it would be interesting to also try and predict target rotational velocity offsets as well as other parameters for the PD controls such stiffnesses and damping coefficients. As long as these are also provided as input to the world model we believe this would be a straightforward extension of our work.

Additionally, the off-policiness of our approach provides interesting benefits for the motion tracking problem domain which we only explored briefly in this work. Making use of a more general world model or efficiently transferring a world model across different domains and task definitions could improve the training time and

make the physics-based character motion tracking more accessible in terms of compute.

Finally, our approach should be equally applicable in other methods of physically-based motion imitation such as those that replace the supervised tracking loss with a discriminator loss [Ho and Ermon 2016; Peng et al. 2021; Xu and Karamouzas 2021].

9 CONCLUSION

In this paper we presented a method for motion tracking of physically simulated characters that uses supervised learning to produce a differentiable world model. This allows control policies to learn optimal actions directly via gradient-descent by minimizing tracking losses over finite horizons. We presented our results on a number of difficult and varied tracking tasks, and provided an in-depth comparison to PPO, evaluating the training time, motion quality, and sensitivity to various hyper-parameter settings and other high level design decisions.

REFERENCES

Emmanuel Bengio, Joelle Pineau, and Doina Precup. 2020. Interference and Generalization in Temporal Difference Learning. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 767–777. http://proceedings.mlr.press/v119/bengio20a.html

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (Nov. 2019), 11 pages. https://doi.org/10.1145/3355089.3356536

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016). arXiv:1606.01540 http://arxiv.org/abs/1606.01540

David F. Brown, Adriano Macchietto, KangKang Yin, and Victor Zordan. 2013. Control of Rotational Dynamics for Ground Behaviors. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Anaheim, California) *(SCA '13)*. ACM, New York, NY, USA, 55–61. https://doi.org/10.1145/2485895.2485906

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-based motion capture imitation with deep reinforcement learning. 1–10. https://doi.org/10.1145/3274247.3274506

Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. 2017. Recurrent Environment Simulators. *CoRR* abs/1704.02254 (2017). arXiv:1704.02254 http://arxiv.org/abs/1704.02254

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 Trans. Graph.* 37, 6, Article 179 (Dec. 2018), 10 pages. https://doi.org/10.1145/3272127.3275048

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. https://doi.org/10.1145/1778765.1781156

Erwin Coumans. 2015. Bullet Physics Simulation. In *ACM SIGGRAPH 2015 Courses* (Los Angeles, California) *(SIGGRAPH '15)*. ACM, New York, NY, USA, Article 7. https://doi.org/10.1145/2776880.2792704

Marco da Silva, Yeuhi Abe, and Jovan Popovic. 2008. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Computer Graphics Forum* 27 (04 2008), 371 – 380. https://doi.org/10.1111/j.1467-8659.2008.01134.x

Marc Peter Deisenroth and Carl Edward Rasmussen. 2011. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (Bellevue, Washington, USA) *(ICML'11)*. Omnipress, Madison, WI, USA, 465–472.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. 2017. Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks. arXiv:1605.07127 [stat.ML]

Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. 2018. Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1–9. https://doi.org/10.1109/IROS.2018.8594448

Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. 2015. Learning Reduced-order Feedback Policies for Motion Skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '15)*. ACM, New York, NY, USA, 83–92. https://doi.org/10.1145/2786784.2786802

Alexey Dosovitskiy and Vladlen Koltun. 2016. Learning to Act by Predicting the Future. *CoRR* abs/1611.01779 (2016). arXiv:1611.01779 http://arxiv.org/abs/1611.01779

Haegwang Eom, Daseong Han, Joseph S. Shin, and Junyong Noh. 2019. Model Predictive Control with a Visuomotor System for Physics-Based Character Animation. *ACM Trans. Graph.* 39, 1, Article 3 (Oct. 2019), 11 pages. https://doi.org/10.1145/3360905

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2492–2515. https://doi.org/10.1111/j.1467-8659.2012.03189.x

F. Sebastin Grassia. 1998. Practical Parameterization of Rotations Using the Exponential Map. *J. Graph. Tools* 3, 3 (March 1998), 29–48. https://doi.org/10.1080/10867651.1998.10487493

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. 1998. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 9–20. https://doi.org/10.1145/280814.280816

David Ha and Jürgen Schmidhuber. 2018. World Models. *CoRR* abs/1803.10122 (2018). arXiv:1803.10122 http://arxiv.org/abs/1803.10122

Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. 2014. Online Motion Synthesis Using Sequential Monte Carlo. *ACM Trans. Graph.* 33, 4, Article 51 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601218

Perttu Hämäläinen, Joose Rajamäki, and C. Karen Liu. 2015. Online Control of Simulated Humanoids Using Particle Belief Propagation. *ACM Trans. Graph.* 34, 4, Article 81 (July 2015), 13 pages. https://doi.org/10.1145/2767002

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust Motion In-Betweening. *ACM Trans. Graph.* 39, 4, Article 60 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392480

Havok. 2021. *Havok Physics*. https://www.havok.com/havok-physics/

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). arXiv:1707.02286 http://arxiv.org/abs/1707.02286

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning that Matters. (09 2017).

Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. 2018. Synthesizing Neural Network Controllers with Probabilistic Model based Reinforcement Learning. *CoRR* abs/1803.02291 (2018). arXiv:1803.02291 http://arxiv.org/abs/1803.02291

Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf

Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Trans. Graph.* 39, 4, Article 53 (July 2020), 13 pages. https://doi.org/10.1145/3386569.3392440

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 Trans. Graph.* 38, 4, Article 74 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322963

Ruizhen Hu, Juzhan Xu, Bin Chen, Minglun Gong, Hao Zhang, and Hui Huang. 2020. TAP-Net: Transport-and-Pack using Reinforcement Learning. *CoRR* abs/2009.01469 (2020). arXiv:2009.01469 https://arxiv.org/abs/2009.01469

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. 2019. Learning agile and dynamic motor skills for legged robots. *CoRR* abs/1901.08652 (2019). arXiv:1901.08652 http://arxiv.org/abs/1901.08652

Sumit Jain and C. Karen Liu. 2011. Modal-space Control for Articulated Characters. *ACM Trans. Graph.* 30, 5, Article 118 (Oct. 2011), 12 pages. https://doi.org/10.1145/2019627.2019637

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems*.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable Muscle-Actuated Human Simulation and Control. *ACM Trans. Graph.* 38, 4, Article 73 (July 2019), 13 pages. https://doi.org/10.1145/3306346.3322972

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 8 pages. https://doi.org/10.1145/1778765.1781155

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

Libin Liu and Jessica Hodgins. August 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Transactions on Graphics* 37, 4 (August 2018).

Libin Liu and Jessica K. Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Transactions on Graphics* 36, 3 (2017).

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. https://doi.org/10.1145/2893476

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving Sampling-based Motion Control. *Comput. Graph. Forum* 34, 2 (May 2015), 415–423. https://doi.org/10.1111/cgf.12571

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.

Lennart Ljung. 1999. *System Identification (2nd Ed.): Theory for the User.* Prentice Hall PTR, USA.

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. 2020. CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion. *CoRR* abs/2005.03288 (2020). arXiv:2005.03288 https://arxiv.org/abs/2005.03288

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum Control for Balance. *ACM Trans. Graph.* 28, 3, Article 80 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531386

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. 2019a. Neural Probabilistic Motor Primitives for Humanoid Control. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJl6TjRcY7

Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. 2019b. Reusable neural skill embeddings for vision-guided whole body movement and object manipulation. *CoRR* abs/1911.06636 (2019). arXiv:1911.06636 http://arxiv.org/abs/1911.06636

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware Nonlinear Control of Dynamic Characters. *ACM Transactions on Graphics* 28, 3 (2009).

Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. 2017. Discovering and Synthesizing Humanoid Climbing Movements. *ACM Trans. Graph.* 36, 4, Article 43 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073707

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. 2017. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *CoRR* abs/1708.02596 (2017). arXiv:1708.02596 http://arxiv.org/abs/1708.02596

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. 2015. Action-Conditional Video Prediction using Deep Networks in Atari Games. *CoRR* abs/1507.08750 (2015). arXiv:1507.08750 http://arxiv.org/abs/1507.08750

Soohwan Park, Hoseok Ryu, Sunmin Lee, and Jehee Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.* 38, 6, Article 205 (Nov. 2019).

Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. 2019. Modeling Human Motion with Quaternion-based Neural Networks. *CoRR* abs/1901.07677 (2019). arXiv:1901.07677 http://arxiv.org/abs/1901.07677

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018a. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201311

Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. 2018b. SFV: Reinforcement Learning of Physical Skills from Videos. *ACM Trans. Graph.* 37, 6, Article 178 (Nov. 2018), 14 pages.

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

PhysX. 2021. *PhysX.* https://developer.nvidia.com/physx-sdk

Marc H. Raibert and Jessica K. Hodgins. 1991. Animation of Dynamic Legged Locomotion. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. Association for Computing Machinery, New York, NY, USA, 349–358. https://doi.org/10.1145/122718.122755

Jürgen Schmidhuber. 1990. *Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments.* Technical Report.

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

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). https://doi.org/10.1145/1276377.1276511

Richard S. Sutton. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.* 2, 4 (July 1991), 160–163. https://doi.org/10.1145/122344.122377

Yuval Tassa, Tom Erez, and Emanuel Todorov. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4906–4913. https://doi.org/10.1109/IROS.2012.6386025

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. 2015. From Pixels to Torques: Policy Learning with Deep Dynamical Models. arXiv:1502.02251 [stat.ML]

Tingwu Wang, Yunrong Guo, Maria Shugrina, and Sanja Fidler. 2020. UniCon: Universal Neural Controller For Physics-based Character Motion. arXiv:2011.15119 [cs.GR]

Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation* 1, 2 (1989), 270–280. https://doi.org/10.1162/neco.1989.1.2.270

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Trans. Graph.* 39, 4, Article 33 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392381

Jungdam Won and Jehee Lee. 2019. Learning Body Shape Variation in Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 207 (Nov. 2019), 12 pages. https://doi.org/10.1145/3355089.3356499

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

Pei Xu and Ioannis Karamouzas. 2021. A GAN-Like Approach for Physics-Based Imitation Learning and Interactive Character Control. *CoRR* abs/2105.10066 (2021). arXiv:2105.10066 https://arxiv.org/abs/2105.10066

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.* 26, 3, Article 105 (July 2007). https://doi.org/10.1145/1276377.1276509

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetry and Low-energy Locomotion. *CoRR* abs/1801.08093 (2018). arXiv:1801.08093 http://arxiv.org/abs/1801.08093

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.* 37, 4, Article 145 (July 2018), 11 pages. https://doi.org/10.1145/3197517.3201366

Bo Zhou, Hongsheng Zeng, Fan Wang, Yunxiang Li, and Hao Tian. 2019. Efficient and Robust Reinforcement Learning with Uncertainty-based Value Expansion. *CoRR* abs/1912.05328 (2019). arXiv:1912.05328 http://arxiv.org/abs/1912.05328