Learning predict-and-simulate policies from unorganized human motion data

SOOHWAN PARK, Seoul National University
HOSEOK RYU, Seoul National University
SEYOUNG LEE, Seoul National University
SUNMIN LEE, Seoul National University
JEHEE LEE, Seoul National University

Fig. 1. Our predict-and-simulate policy creates an agile, interactively-controllable, physically-simulated character equipped with various motor skills learned from unorganized motion data.

The goal of this research is to create physically simulated biped characters equipped with a rich repertoire of motor skills. The user can control the characters interactively by modulating their control objectives. The characters can interact physically with each other and with the environment. We present a novel network-based algorithm that learns control policies from unorganized, minimally-labeled human motion data. The network architecture for interactive character animation incorporates an RNN-based motion generator into a DRL-based controller for physics simulation and control. The motion generator guides forward dynamics simulation by feeding a sequence of future motion frames to track. The rich future prediction facilitates policy learning from large training data sets. We will demonstrate the effectiveness of our approach with biped characters that learn a variety of dynamic motor skills from large, unorganized data and react to unexpected perturbation beyond the scope of the training data.

CCS Concepts: • **Computing methodologies** → **Animation**; **Physical simulation**.

Additional Key Words and Phrases: Interactive character animation, Physics-based Simulation, Data-driven animation, Deep reinforcement learning, Recurrent neural networks

ACM Reference Format:
Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.* 38, 6, Article 205 (November 2019), 11 pages. https://doi.org/10.1145/3355089.3356501

Authors' addresses: Soohwan Park, Department of Computer Science and Engineering, Seoul National University, shpark@mrl.snu.ac.kr; Hoseok Ryu, Department of Computer Science and Engineering, Seoul National University, hoseok.ryu@mrl.snu.ac.kr; Seyoung Lee, Department of Computer Science and Engineering, Seoul National University, seyounglee@mrl.snu.ac.kr; Sunmin Lee, Department of Computer Science and Engineering, Seoul National University, sunmin.lee@mrl.snu.ac.kr; Jehee Lee, Department of Computer Science and Engineering, Seoul National University, jehee@mrl.snu.ac.kr.

1 INTRODUCTION

Creating a virtual character from a collection of human motion data has been a long-standing problem in computer graphics. The motion data set serves as a source of plausible actions and behavioral patterns available to the character. It is challenging to understand what actions are available from unorganized motion data sets, how the actions generalize, and how to generate a sequence of plausible actions to animate a virtual character. The problem becomes even more challenging if we want the character to be physically simulated and interactively controlled to perform task-driven actions while maintaining its balance under gravity.

The recent advancement of Deep Reinforcement Learning (DRL) has made drastic improvements in physically based biped control. It has been demonstrated that the control policy of an under-actuated dynamics system can be effectively learned and compactly represented by deep networks. DRL approaches can be classified into three categories. The first category includes uninformed DRL algorithms that learn control policies from scratch without any example or reference data [Heess et al. 2017]. The second category includes imitation learning algorithms that take reference motion data as input and control the character to mimic the reference data [Peng et al. 2018]. The algorithms in the third category exploit a collection of motion data to learn a diversity of human actions and transitions between the actions in a single control policy. The algorithms in the first category are ideal if reference data are unavailable. For example, motion data are unavailable or difficult to acquire if the character is non-anthropomorphic or imaginary. From the graphics point of view, the best-looking results are often produced by imitation learning with a short clip of reference data. We are particularly interested

in the third category, where the motion data set is unorganized, unlabeled, and has rich variability in the action repertoire.

In this paper, we will present a novel network-based algorithm that learns a multi-objective control policy from unorganized motion data, which may include various actions and styles driven by different tasks or objectives. The character is equipped with a rich repertoire of motor skills and physically simulated driven by the control policy. The user can control the character interactively by modulating control objectives (e.g., walk towards a target direction at a certain speed and move to a target location in a certain time). The character is resilient against external perturbation and can interact physically with the environment. This problem poses several technical challenges. First, the system should be able to generate a conditioned sequence of human-like, physically-plausible actions from unorganized, minimally-labeled motion data. Second, the control policy should be learned from a large set of training data that can accommodate rich variability in human motion. Lastly, the character should be able to physically interact with the environment and other characters. It is very difficult, if not impossible, to capture all possible physical interactions between full-body characters in the training data. So, the character has to learn how to react to unexpected perturbation beyond the scope of the training data.

To address the issues, we present a novel architecture for interactive character animation that incorporates an RNN-based motion generator into a DRL-based controller for physics simulation and control. The motion generator guides forward dynamics simulation by feeding a sequence of future motion frames and also learns physical plausibility from forward dynamics simulation and the control policy. The rich future prediction facilitates policy learning from large training data sets. We will demonstrate that our *predict-and-simulate* framework makes it possible to learn robust control policies from large, unorganized training data.

2 RELATED WORK

2.1 Data-Driven Animation

Creating an interactively-controllable character from a collection of unorganized motion capture data has been a fundamental goal of data-driven animation. A classical approach constructs a motion graph from the data set [Lee et al. 2002]. The motion graph includes a collection of short motion segments connected to each other. Given a user-specified control objective, state-space search through the motion graph reshuffles motion segments and consequently generates a novel sequence of motion. The concept of reinforcement learning was introduced to the computer graphics community by Lee et al.[2004] to pre-compute state-space search on the motion graph and tabulate the resultant control policies for interactive character animation. This approach has further been improved by employing linear function approximators, which encodes the control policies memory-efficiently [Treuille et al. 2007]. Motion synthesis based on the motion graph is discrete and combinatorial because short motion segments serve as basic building blocks in the synthesis process. Alternatively, motion synthesis in continuous space through low-dimensional embeddings of motion data has also been explored by many researchers [Grochow et al. 2004; Lee et al. 2010b; Levine et al. 2012; Shin and Lee 2006]. In general, interactive motion synthesis is a complex process that deals with both combinatorial sequencing in discrete action space and their spatiotemporal variations in continuous space [Hyun et al. 2016; Min and Chai 2012; Won et al. 2017].

The advancement of deep learning impacted data-driven animation recently. Holden et al. [2016] stacked a deep feedforward neural network on top of the trained autoencoder to animate the character to walk along a curved trajectory and punch/kick at a target. In their follow-up study, they built a *Phase-Functioned Neural Network* to learn the motion repertoire of the character traversing uneven terrain [Holden et al. 2017]. The *Mode-Adaptive Neural Network* by Zhang et al. [2018] exploited a mixture of experts model to learn quadruped locomotion using feedforward neural networks. Alternatively, there has been a stream of research that exploits recurrent neural networks [Fragkiadaki et al. 2015; Ghosh et al. 2017; Harvey and Pal 2018; Zhou et al. 2018]. Lee et al. [2018b] constructed an RNN-based framework for interactive character animation and discussed the concept of multi-objective control. The feedforward networks and recurrent networks are all learned in a supervised manner. Since deep networks are adept at approximating the behavior of any complex function, it is known that the representation power of deep networks can cope with both combinatorial and continuous variations simultaneously from the training data.

2.2 Physically Based Animation

Designing a physics-based controller for biped locomotion has long been a notoriously challenging problem for decades in computer graphics and robotics. Early (before the advent of deep reinforcement learning) approaches made use of hand-craft feedback rules [Ha et al. 2012; Yin et al. 2007], simplified dynamics models (e.g., inverted pendulum [Kwon and Hodgins 2010; Tsai et al. 2010] and pendulum-on-cart [Kwon and Hodgins 2017]), data-driven control with motion capture references [Lee et al. 2010a], stochastic optimal control [Al Borno et al. 2013; Barbič et al. 2009; Ha and Liu 2014; Liu et al. 2016, 2012; Mordatch et al. 2012; Tan et al. 2014; Wang et al. 2010, 2012], and model predictive control [Hämäläinen et al. 2015; Tassa et al. 2012, 2014]. Sok et al. [2007] learned biped controllers from a small collection of motion capture data using trajectory optimization and locally-weighted linear regression. A similar local regression method was used to learn the flapping controller from the motion capture of flying birds [Ju et al. 2013]. Since the representation power of linear regression functions is limited, the linear function can approximate only a small portion of the training data. This limitation can be overcome by using more powerful regression functions, such as deep networks.

Deep reinforcement learning has been very successful in many control problems [Bansal et al. 2017; Heess et al. 2017]. Peng et al. [2017] built a hierarchical RL architecture that provides control over the character to meet user-specified goals. The high-level controller generates footsteps to achieve the goals at coarse time-scales, while the low-level controller directly actuates joints to move the full-body and follow the footsteps at fine time-scales. In the follow-up study [Peng et al. 2018], control policies are constructed to imitate the dynamic behaviors captured in short motion clips. Liu et al. [2017] used deep Q-learning to schedule control fragments.

Each control fragment allows the character to perform a short motor skill. They also constructed the control policy of basketball dribbling skills using trajectory optimization and DRL [Liu and Hodgins 2018]. Flying creatures have also been simulated and controlled using DRL algorithms that are enhanced by evolutionary exploration [Won et al. 2017] and self-regulated learning [Won et al. 2018]. Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon 2016; Merel et al. 2017] is an alternative approach that learns an imitation policy in a manner similar to Generative Adversarial Networks (GAN) [Goodfellow et al. 2014]. A classifier is trained to discriminate motion capture demonstrations from the imitation data, while the imitation policy is given reward for fooling the discriminator. GAIL approaches aim at constructing more robust controllers from fewer demonstrations with exceptions [Wang et al. 2017], which aims at learning diverse behaviors.

2.3 Combining Data-Driven and Physically Based Animation

There have been continual efforts to combine the interactivity and flexibility of data-driven animation with the responsiveness and plausibility of physically based simulation and control. Data-driven methods and physically based methods can interact with and reinforce each other in several ways. The motion capture data are often used to guide physics-based simulation [Chentanez et al. 2018; Lee et al. 2019, 2010a; Merel et al. 2018; Peng et al. 2018]. Furthermore, interactively controlling physics-based character in real-time is attainable by combining a motion matching technique and DRL-based control policy [Bergamin et al. 2019]. Conversely, physics laws allow motion data to be manipulated and edited in a physically plausible manner [Sok et al. 2010; Ye and Liu 2010]. The characters in data-driven animation are often encumbered to stay on the course of observed actions or their limited generalization. To alleviate this limitation, the character can be allowed to deviate from the course to react to external pushes by adapting the motion data [Zordan and Hodgins 2002]. Alternatively, it is also possible to switch between data-driven and physics simulation modes so that the character is physically simulated between example motion data trajectories to react to external hit and push [Zordan et al. 2007]. Whereas transitioning from data to simulation is straightforward, transitioning back from simulation to data is a challenging problem that requires searching through a database of motions. Our system also combines data-driven modules (e.g., RNN-based motion generator) with physics-based modules (e.g., DRL-based control policy). They continuously interact with each other to accentuate individual strengths and address their limitations.

3 INTERACTIVE CHARACTER ANIMATION

We consider a humanoid character that takes a control command from the user and makes a move by actuating its joints. The character is capable of performing various actions, and each individual action may take different control objectives. We adopt the concept of multi-objective control to formulate the framework of interactive character control [Lee et al. 2018b]. The objective of control is given as a tuple $C = (\alpha, \{\beta^i\})$ that includes an action label α, and a list of continuous control objectives $\{\beta^i\}$ chosen from many possible ones, such as target direction, speed, position, timing, and target height of COM (Center of Mass). The control objective has either a maintenance goal or an achievement goal. For example, commanding the character to walk continuously towards direction $v \in R^2$ at speed 0.3 meter/second is a maintenance goal described by (Walk, $\{v, 0.3\}$). Commanding the character to punch at a target $p \in R^2$ in 2 seconds is an achievement goal (Punch, $\{p, 2\}$).

It is not necessary for the user to specify values for all control objectives all the time. For example, the moving direction and the speed are relevant parameters while the character is walking. If the character falls over, the height of COM becomes a key control objective that commands the character to stand up while the moving direction and the speed become less relevant. The user can leave irrelevant parameters unspecified.

Our system consists of two main components: The motion generator and the tracking controller (see Figure 2). The motion generator is a discrete-time model that sequentially generates the kinematic motion of the character conditioned by control objectives. Let $\mathbf{s}_t$ be the current state of the character at time t. The motion generator is a network-based recurrent model

$$(\hat{P}_{t+1}, \hat{C}_{t+1}) = G^*(\mathbf{s}_t, C_t). \tag{1}$$

Given a control objective C_t, the output of the motion generator includes the prediction $\hat{P}$ of the full-body poses of the character and its location at forthcoming time steps, which serves as a reference guide for physics-based simulation (see Figure 3). The prediction $\hat{C}$ of control objectives is used to fill in unspecified control objectives at the next time step. The tracking controller runs forward dynamics simulation to produce the actual state at time $t + 1$ driven by the control policy

$$\mathbf{s}_{t+1} = \text{Simulation}\big(\mathbf{s}_t, \pi(\mathbf{a}|\mathbf{s}_t, \hat{P}_{t+1})\big). \tag{2}$$

The control policy learned through DRL generates appropriate actuation at the character's joints to track the reference guide while maintaining its balance under gravity. The trace of forward dynamics simulation is fed back into the motion generator for the prediction of the next motion.

This architecture has technical advantages addressing the limitations of data-driven and physically based animation. From the viewpoint of data-driven animation, the robust control policy from DRL allows the output of the motion generator to be fed into forward dynamics simulation and thus the character can physically interact with the environment. From the viewpoint of physically based animation, the interactivity and controllability of data-driven animation allow us to have direct control over the action of physically-simulated characters, which can be controlled with spatiotemporal goals and continuous control objectives rather than simply controlled to trace a pre-defined course of action. Physical interaction and response allow the character to deviate from the scope of action repertoires captured in the training data. The generalization capability of data-driven modeling makes it possible for the character to come back to the regular routine of data-driven animation after deviation. Consequently, we can have direct control over the character against unexpected occurrences of physical events (e.g., falling down, tripping over, the response against push and hit) not recorded in the training data.

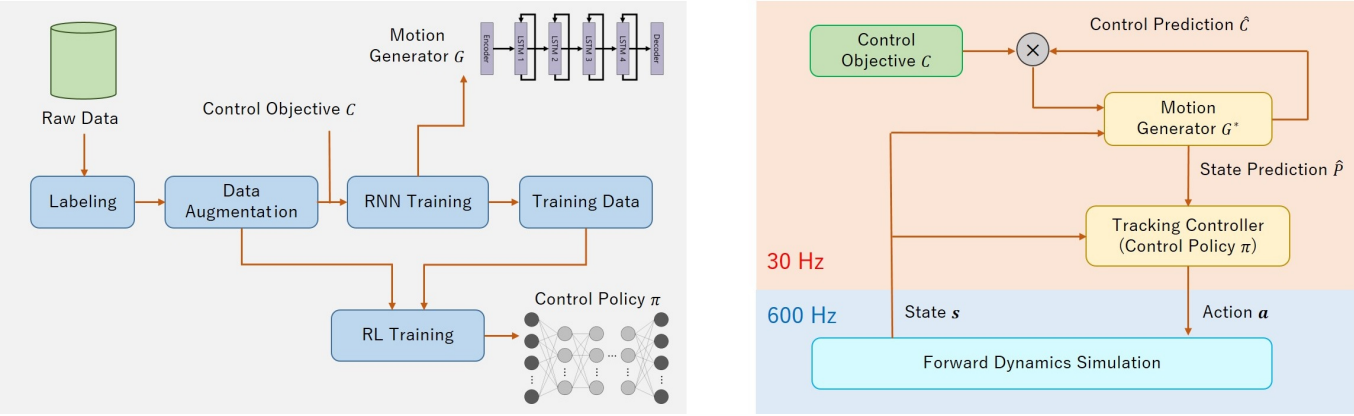

Fig. 2. System overview for interactive character animation. In the preprocessing phase (left in the figure), the system learns motion generator G and control policy π from the training data, which are integrated into the system for runtime simulation (right in the figure).

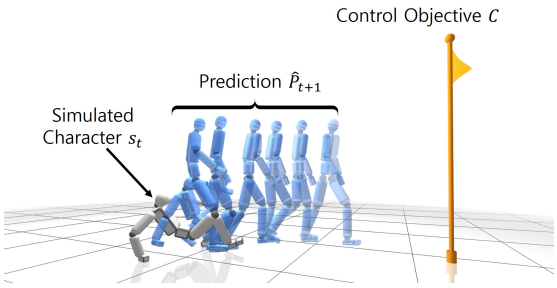

Fig. 3. The state, the prediction and the control objective.

4 MOTION GENERATOR

Our motion generator is a recurrent model of human activities learned from a collection of human motion data. Its key functionality is to predict future motion given a control objective. The per-frame generator G is defined by

$$(\hat{\mathbf{s}}_{t+1}, \hat{C}_{t+1}) = G(\mathbf{s}_t, C_t), \tag{3}$$

which takes the current state as input and outputs the prediction for the next time step. The per-frame generator consists of multiple layers of LSTM (Long Short-Term Memory) units with encoder and decoder layers. The state includes the position and orientation of the skeletal root, angles at all joints, 3D positions of all joints and a binary vector encoding whether individual body parts, such as forefeet and heels, are in contact with the ground surface. The description of the state is redundant because it includes both joint angles and joint positions simultaneously. This redundancy is useful for data-driven learning of articulated figure motion as reported in previous studies [Holden et al. 2017; Lee et al. 2018b; Zhang et al. 2018].

The motion generator G^* in equation (1) is defined by applying the per-frame generator recursively to produce the prediction

$$\hat{P}_{t+1} = (\hat{\mathbf{q}}_{t+\tau_1}, \hat{\mathbf{v}}_{t+\tau_1}, \cdots, \hat{\mathbf{q}}_{t+\tau_n}, \hat{\mathbf{v}}_{t+\tau_n}), \tag{4}$$

where τ_i are temporal offsets, $\hat{\mathbf{q}}_{t+\tau_i}$ are pose predictions, and $\hat{\mathbf{v}}_{t+\tau_i} = (\hat{\mathbf{q}}_{t+\tau_i+1} - \hat{\mathbf{q}}_{t+\tau_i})/\Delta t$ are velocity predictions, where Δt is the time step size (1/30 seconds). The pose and velocity predictions are designed to match the dynamic state of the character. The first temporal offset is one time step ahead and the subsequent temporal offsets are sampled sparser than the control frame rates.

4.1 Motion Data Processing

The per-frame generator is learned from training data sets that can be as small as a few motion clips of less than ten seconds in playtime and as large as several hours. We label the training data sets with action types. A single motion frame may have multiple action labels. Fortunately, our learning algorithm is not sensitive to the exact boundary of labels, so the manual process is not laborious. Foot-ground contacts can be detected and labeled automatically from motion capture data to indicate when the right and left foot touch the ground.

Training data sets often lack combinatorial variations (e.g., sequential transitions between actions) and spatiotemporal variations (e.g., turning angles and walking speed). Rich combinatorial variations make the character more responsive, while rich spatiotemporal variations allow the character to follow the command more precisely. In order to enrich combinatorial variabilities, we first construct a motion graph [Lee et al. 2002] from the training data and generate a sequence of motions via a random walk through the graph. Random walk reshuffles motion frames in the data set and thus generates a large variety of sequential combinations of actions. We further process the random data to enrich spatial and temporal variations by using Laplacian editing [Kim et al. 2009]. Given a random sequence of motion, we divide it into segments of random lengths (2 to 5 seconds) and lengthens/shortens/bends the spatial trajectory of each individual segments randomly within the range of its length from 70% to 130% and its angle from $-45°$ to $45°$. We concatenate all modified segments back together to generate the augmented training data, which is denoted by $\{\mathbf{m}_t\}$. $\mathbf{m}_t$ is the full-body pose of the character at frame t.

In our experiments, we generated a sufficiently large (ten minutes to five hours) set of augmented data for each example depending on the size of the original data set and the complexity of the tasks. The

redundancy of the training data does not affect the performance of network training because the learning process requires a lot of iterations until it plateaus and will consume the data repeatedly if the data set is small.

4.2 Generator Network Training

The per-frame generator is trained through supervised learning. The motion capture data records what the subject performs, but his/her intention is not recorded in the data set. Understanding the intention of taking action is important because it clarifies whether the action serves for achieving a particular control objective. The subject might have a multitude of intentions while performing an action. We infer control objectives at every frame of the motion data as follows. Let $C = (\alpha, \{\beta^i\})$ be a template of control objectives we want to infer from the data set. If action α has a maintenance goal, for every motion frame $\mathbf{m}_t$ with the action label, we assume that the character at frame t has intention to perform the action with parameters β^i (e.g., average moving direction and average speed) estimated in the interval $[t + t_{\min}, t + t_{\max}]$. If the action α has an achievement goal, for every motion frame $\mathbf{m}_t$ with the action label, we assume that the character at frame $t - \tau$, for $\tau < T$, has the intention to perform the action in time τ at the position and orientation indicated by $\mathbf{m}_t$. Here, T is the duration of time the character looks ahead. In our experiments, we set $t_{\min} = 0.5s$, $t_{\max} = 1.0s$ and $T = 3.0s$.

Given a sequence of states $\mathbf{s}_t$ and inferred intentions C_t in the training data, supervised learning minimizes the loss function including four terms.

$$E = w_1 E_{\text{motion}} + w_2 E_{\text{objective}} + w_3 E_{\text{contact}} + w_4 E_{\text{rigid}}. \tag{5}$$

E_{motion} and $E_{\text{objective}}$ penalize the discrepancy between training data and network outputs.

$$\begin{aligned} E_{\text{motion}} &= \sum_t \|\mathbf{s}_{t+1} - \hat{\mathbf{s}}_{t+1}\|^2, \\ E_{\text{objective}} &= \sum_t \sum_i \|\beta^i_{t+1} - \hat{\beta}^i_{t+1}\|^2, \end{aligned} \tag{6}$$

where $(\hat{\mathbf{s}}_{t+1}, \hat{C}_{t+1}) = G(\mathbf{s}_t, C_t)$ be the output of the network, $C_t = (\alpha_t, \{\beta^i_t\})$ and $\hat{C}_t = (\hat{\alpha}_t, \{\hat{\beta}^i_t\})$. The hat symbol indicates the quantities generated by the network. E_{contact} penalizes foot sliding.

$$E_{\text{contact}} = \sum_t \sum_{k \in \text{Feet}} \hat{c}^k_t \hat{c}^k_{t+1} \|\hat{h}^k_t - \hat{h}^k_{t+1}\|^2, \tag{7}$$

where $\hat{h}^k$ is the position of the k-th joint in the reference coordinate system. Here, $\hat{c}^k_t = 1$ if joint k is in contact with the ground surface at time step t. Otherwise, $\hat{c}^k_t = 0$. The condition $\hat{c}^k_t = \hat{c}^k_{t+1} = 1$ indicates that the contact remains for the time step and thus the joint should not slide. E_{rigid} penalizes the lengthening and shortening of body links.

$$E_{\text{rigid}} = \sum_t \sum_{(k,m) \in \text{Links}} (\|\hat{h}^k_t - \hat{h}^m_t\| - l_{km})^2, \tag{8}$$

where joint k and joint m are adjacent to each other connected by a rigid link and l_{km} is the length of the link. The distance between the adjacent pair of joints should be preserved. In our experiments, we set $w_1 = 1, w_2 = 0.3, w_3 = 6$ and $w_4 = 0.01$. Given the loss function, the multi-layer encoder-LSTMs-decoder network is learned using backpropagation through time and layers.

Our implementation of the motion generator is largely based on the work of Lee et al. [2018b] with a key difference that our motion generator takes dynamics states from the physics simulator as input rather than feeding the output of the network directly into the input for the next time step. This simple change allows a lot of flexibility in network learning and applications. Most importantly, this setup allows the motion generator to deviate from the training data, react to external perturbation, and physically interact with the environment.

5 TRACKING CONTROLLER

Example-guided approaches have been successful in physically based character animation [Lee et al. 2010a; Peng et al. 2018]. Given a short motion clip, the example-guided control policy produces appropriate actuation at joints to imitate the reference motion. The target state is often simplified to a phase variable in the duration of the reference motion data. This approach with DRL algorithms is capable of reproducing high-quality human movements in physics simulation. However, the performance of the algorithm and the quality of the results degrade as the size of the reference data increases.

The functionality of our tracking controller is similar to example-guided control policies in the sense that it takes a stream of motion frames as input and physically simulates the biped character to reproduce the input stream. Our tracking controller addresses two fundamental problems of example-guided approaches. First, the input stream is not fixed, but it may change unpredictably. The motion generator produces a stream of motion frames in accordance with the user's interactive control and thus the prediction of future frames may change dynamically. The tracking controller should be able to cope with interactive changes at the run-time simulation. Secondly, learning a control policy from a large variety of unorganized motor skills is a technical challenge. To address this issue, we come up with two ideas. We found that the predictive information produced by the motion generator improves the discriminative power of RL control policies to deal with rich combinatorial and spatiotemporal variations. We also found that adaptive sampling of initial state distribution improves the overall performance by exploring easy-to-learn motor skills and difficult-to-learn motor skills in a balanced manner.

5.1 DRL formulation

The goal of reinforcement learning is to find the parameters θ^* for the optimal policy $\pi_{\theta^*}(\mathbf{a}|\mathbf{s})$ that maximizes the expected cumulative reward.

$$\theta^* = \underset{\theta}{\text{argmax}}\, \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \Big[\sum_t \gamma^t r_t \Big] \tag{9}$$

where $\mathbf{s}_t$, $\mathbf{a}_t$ and r_t are the state, action and reward at time t, $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is a dynamics simulator that generates the character's next pose from the current pose and the action it takes during the time step. $\gamma \in [0, 1]$ is a discount factor.

The *state* of the motion generator network was designed to represent the kinematics of the full-body, including joint positions and

joint angles. The state of the policy network includes additional features to represent the full-body dynamics effectively. To distinguish the dynamic state from the kinematic state, we denote the input to the policy network by $\mathbf{s}_{\text{dynamic}}$. The dynamic state $\mathbf{s}_{\text{dynamic}}$ includes the generalized coordinates of the full-body pose, its generalized velocity, the up-vector θ_{up} of the trunk, the end-effector positions, the height of the skeletal root and the ground clearance at both feet.

The *action* $\mathbf{a}$ is defined by a target pose of PD controllers. The future prediction at the next time step serves as a target pose $\hat{\mathbf{q}}$ to track. The action of RL modulates the predicted target to control the full-body motion precisely.

$$\mathbf{q}_{\text{d}} = \hat{\mathbf{q}} + \Delta\mathbf{q}_{\text{d}}, \tag{10}$$

where $\mathbf{q}_{\text{d}}$ is the PD target and $\Delta\mathbf{q}_{\text{d}}$ is the output (action) of the control policy. We use stable PD control [Tan et al. 2011] for improved stability.

The goal of the control policy is to track the future prediction $\hat{P}$ reliably at every frame of the simulation. The *reward* is set to meet this goal.

$$r = r_q \cdot r_v \cdot r_{\text{e}} \cdot r_{\text{com}}, \tag{11}$$

where pose, velocity, end-effector, and COM matching rewards respectively are

$$r_q = \exp\Big(-\frac{1}{\sigma_q^2} \sum_i \| \hat{q}_i \ominus q_i \|^2 \Big)$$

$$r_v = \exp\Big(-\frac{1}{\sigma_v^2} \sum_i \| \hat{v}_i - v_i \|^2 \Big)$$

$$r_{\text{e}} = \exp\Big(-\frac{1}{\sigma_{\text{e}}^2} \sum_j \| \hat{p}_j - p_j \|^2 \Big)$$

$$r_{\text{com}} = \exp\Big(-\frac{1}{\sigma_{\text{com}}^2} \| \hat{p}_{\text{com}} - p_{\text{com}} \|^2 \Big)$$

Here, q_i are joint orientations described by quaternions, v_i are joint angular velocities, p_j are end-effector (left and right feet) positions, and p_{com} is the center of mass position. The hat symbol indicates desired values in the prediction. i and j are the indices of joints and end-effectors. In our implementation, $\sigma_q = 14.4, \sigma_v = 72, \sigma_{\text{e}} = 9.6$ and $\sigma_{\text{com}} = 0.6$. We multiply reward terms to combine them rather than adding them with individual weights. As discussed by Lee et al. [2019], the multiplication of the terms gets rewards only when all conditions are met simultaneously. In our experiments, the multiplicative composition produces more accurate control policies at increased computational costs.

5.2 Value and Policy Network Training

We use two sets of motion trajectories for training. The augmented motion data set we used to learn the motion generator is also employed for training the policy and value functions. The motion data set provides us with available ranges of control objectives, such as the ranges of walking speeds and steering directions. We can generate a plausible sequence of random control inputs within the ranges, from which the motion generator produces the second set of synthetic motion trajectories. The benefits of using the synthetic data set for RL are twofold. First, the synthetic data set enriches the correlation between control inputs and resultant actions. Secondly, the quality and distribution of the augmented data set are similar to those of the original data set. Though the motion generator is also supposed to preserve the characteristics of the training data as much as possible, this goal is not perfectly satisfied in practice. The RL policies are expected to learn the characteristics of the motion generator from the synthetic data set.

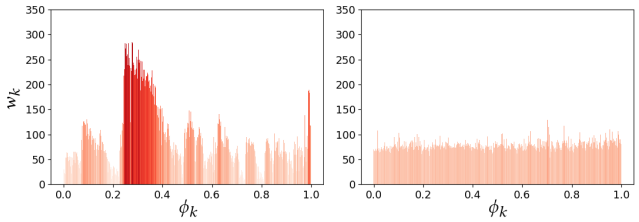

Fig. 4. Uniform vs adaptive sampling of initial states. (Left) The average cumulative reward starting at a segment is influenced by the difficulty of learning upcoming motor skills. (Right) Adaptive sampling provides uniform chances to improve motor skills through out the entire data set.

The policy π_θ and value function V_ψ are modeled as fully-connected neural networks, where θ and ψ are network parameters. The policy network π_θ maps a state $\mathbf{s}_{\text{dynamic}}$ and prediction $\hat{P}$ to a Gaussian distribution over action

$$\pi_\theta(\mathbf{a}|\mathbf{s}_{\text{dynamic}}, \hat{P}) = \mathcal{N}(\mu(\mathbf{s}_{\text{dynamic}}, \hat{P}), \text{diag}(\sigma)) \tag{12}$$

with a diagonal covariance matrix $\text{diag}(\sigma)$. The sizes of the input and output layers are the dimension of states and actions, respectively. We adopt proximal policy optimization [Schulman et al. 2017] and generalized advantage estimation [Schulman et al. 2015] to learn the policy and value functions. The value function is updated using target values computed with TD(λ) [Sutton and Barto 1998].

The learning algorithm is episodic. It collects a batch of experience tuples from many episodic simulations. Each episode starts with an initial state $\mathbf{s}_0$ randomly chosen from the training trajectory. Rollouts are generated by sampling actions from the policy at every step. Each episode terminates when it reaches the end of the training trajectory or a termination condition is met (e.g., falling over or deviating from the trajectory beyond a user-specified threshold).

5.3 Adaptive Sampling of Initial States

The motion data set includes many motor skills performed in an unorganized manner. Some of the motor skills are easy-to-learn, while some are more difficult-to-learn. This leads to an unbalanced sampling problem. Peng et al. [2018] proposed random state initialization (RSI) to draw initial states uniformly from the duration of the reference motion assuming that the duration is reasonable short. The goal of adaptive sampling is to draw more samples from where it needs improvements and less samples from where it already mastered skills (see Figure 4). The adaptive sampling is more effective than uniform sampling with larger training data.

We consider the whole trajectory in the training data parameterized by $[0, 1]$ and divide the trajectory uniformly into n segments, where $\phi_k = [\frac{k}{n}, \frac{k+1}{n})$. Let w_k be the average cumulative rewards of all episodes starting in segment ϕ_k.

$$w_k = \mathbb{E}_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t),\mathbf{s}_0\sim U(\phi_k)}\Big[\sum_t r_t\Big] \tag{13}$$

Table 1. The size of the original and augmented data sets. The size is measured by frames. The motion data are sampled at 30 frames per second.

Example	Original data	Augmented data
zombie	950	10,000
gorilla	342	10,000
chicken hopping	425	10,000
walk, run and stand up	1,845	20,000
obstacle racing	1,816	110,000
fight	4,777	110,000
basketball	21,600	490,000

Whenever a new episode begins, we select a random segment with Boltzmann weight

$$\rho_A(\phi_k) = \frac{exp(-w_k/\upsilon)}{\sum_i exp(-w_i/\upsilon)}, \tag{14}$$

where υ is a parameter for regulating the difference in probabilities according to the difference in average cumulative rewards. In our implementation, we use a constant value for $\upsilon = 5$. The initial state $\mathbf{s}_0$ of the new episode is selected uniformly in segment ϕ_k. The weights at the end of the trajectory needs normalization because an episode starting at the end will have less chance to cumulate rewards than other episodes started earlier.

6 EXPERIMENTS

The simulator is written in C++ with DART open source dynamics toolkit [Lee et al. 2018a]. The neural network operations and the reinforcement learning algorithms are written in Python with Tensorflow [Abadi et al. 2016]. The skeleton of the character has 17 ball-and-socket joints and is controlled by applying joint torques at the joints. The character is 1.5 meter tall and weighs 49 kg. The joint torques are computed by SPD (Stable Proportional Derivative) controller [Tan et al. 2011]. The PD gains are $k_p = 1,000$ and $k_d = 100$ for the hip joints and $k_p = 500$ and $k_d = 50$ for the other joints. The motion generator consists of four LSTM layers with 512 nodes. The time step for truncated BPTT is 48. The policy network has four fully-connected layers with 1024 ReLU nodes and the value network has two fully-connected layers with 512 ReLU nodes. The clip range of PPO is 0.2, the learning rates are 2e-4 for the policy function and 1e-3 for the value function, respectively. $\gamma = 0.95$ to 0.99, and $\lambda = 0.95$. The optimization of the network is executed after 20,000 transition tuples are collected. The minibatch size is 1024. The sizes of the motion clips and the augmented data sets are shown in Table 1. We collected motion data from CMU motion databases, SNU motion databases, and Motionbuilder example sets. All motion data are sampled at 30 frames per second. The forward dynamics simulation proceeds at the rate of 600 frames per second. Learning a motion generator takes 12 to 24 hours and learning a tracking controller takes 24 to 96 hours on a single PC with Intel i9-9900k (3.6GHz, 16 cores) and NVidia Geforce RTX 2070.

6.1 Learning from Small Data

Interactive characters, *zombie* and *gorilla*, can be trained from a small motion data set (see Figure 5). The zombie learned its control policy from a few motion clips of 31.7 seconds playtime including several steps of straight walking and turning. The gorilla learned its control policy from a 10.4 second data set including side steps and roaring. Even with these small data sets, the data augmentation algorithm enriches combinatorial and spatiotemporal variabilities to a significant degree. There is only one action id, MoveTo and the target position $\{\beta_x, \beta_z\}$ is provided. Their lookahead time for control inference is 1.0 second. The target position is feasible if the character can reach the position in the lookahead time. The feasible range (the angle of moving directions and the distance to the target) of the target position estimated from the augmented data set is much broader than the range estimated from the original motion clips. At the run-time simulation, user-specified target positions outside the feasible range are projected into the range for stable physics-based tracking.

Chicken hopping is a game of hopping on one leg and bumping the opponent to fall him/her over. The character learned to play the game from a motion data set of only 14 seconds. The learned controller allows the character to move in any direction and turn while hopping continuously. The control parameters are identical to the previous examples. In the multi-player scene, two teams play the game with three players in each team. Each player moves to the nearest opponent and bumps while maintaining its balance against the impulse (see Figure 5).

6.2 Physical Response

The pedestrian character learned to walk and run from a one minute data set, which also includes the motion of falling forward/backward and standing up again. In this example, *walk, run and stand up*, the control parameters include five action types and three continuous parameters.

$$C = (\alpha, \{\beta_x, \beta_z, \beta_h\}), \tag{15}$$

where $\alpha = \{\mathsf{Walk} \mid \mathsf{Run} \mid \mathsf{StandUp} \mid \mathsf{FallForward} \mid \mathsf{FallBackward}\}$, (β_x, β_z) is the target position and β_h is the height of the root segment. Among the five action types, the character can choose from the first three actions to control and the other two actions can be taken only when the character loses its balance by an external perturbation. Parameter β_h is the key parameter that drives the fall and stand actions.

If the character deviates from the prediction of the motion generator, its tracking controller loses its control over the body and the system intercepts the control. The system first decreases the PD gains to make the body compliant to an external perturbation and then switches the action type (either FallForward or FallBackward) to deal with the situation gracefully. The fall actions guide the falling body to a pre-defined pose lying on the ground, from which the subsequent StandUp action starts. This feedback architecture allows the character to respond to physical interactions.

In the *obstacle racing* example, the user interactively controls the character rushing through various obstacles. The character is equipped with running, jumping and rolling motor skills. The control parameter is defined by

$$C = (\alpha, \{\beta_x, \beta_z, \beta_h, \beta_t\}), \tag{16}$$

where $\alpha = \{\mathsf{Run} \mid \mathsf{Jump} \mid \mathsf{Roll}\}$ and β_t is the timing of action. The lookahead times for running, jumping and rolling are 0.5, 1.33 and 1.33 seconds, respectively. β_t is initially set to the lookahead time

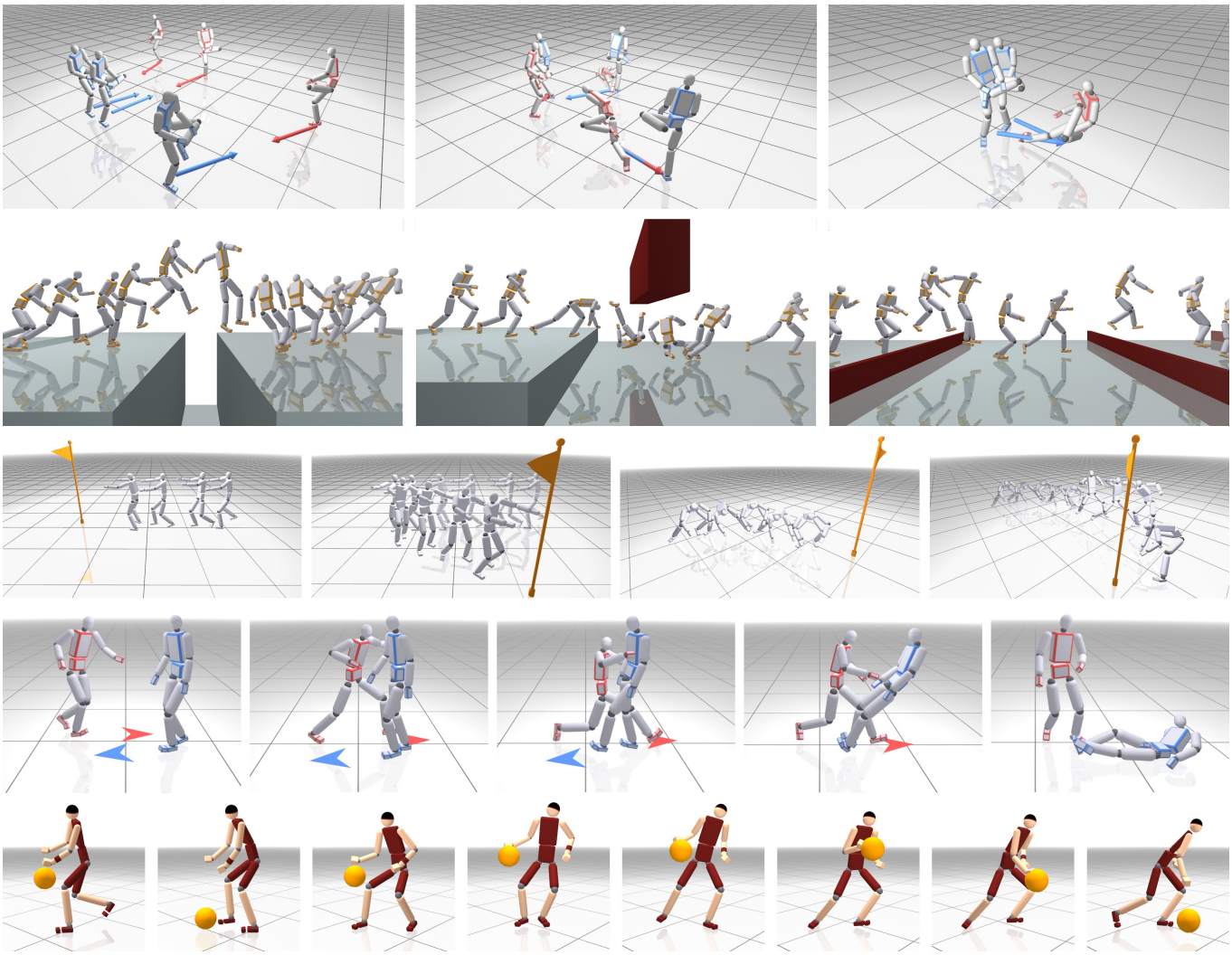

Fig. 5. Screen shots of examples (top to bottom) chicken hopping, obstacle racing, the zombie and the gorilla, fighting, and basketball dribbling.

when the action is triggered by the user and the time to action decreases gradually as the simulation proceeds.

In the *fight* example, the user controls two characters to have them punch and kick each other. The two characters are simulated and controlled by the same motion generator and the same tracking controller. The control parameters are

$$C = (\alpha, \{\beta_x, \beta_z, \beta_t\}), \tag{17}$$

where α = {MoveTo | Punch | Kick | FallForward | FallBackward | StandUp}. The semantics of the continuous parameters may vary depending on the activated action. (β_x, β_z) indicates the target position the character is heading to when MoveTo is activated. When the punch or the kick is triggered, (β_x, β_z) is the target position on the opponent's body to which the end-effector (the fist or the foot) is heading. While falling over, the character is supposed to lie down on the ground at position (β_x, β_z). We temporally decrease the PD gains of the character hit by a punch or a kick so that it can respond to the impact compliantly, analogous to the method suggested by Hämäläinen et al. [2014], which reacts to impact by lowering the maximum joint actuation torques.

6.3 Basketball Dribbling

Our basketball character can dribble the ball while both the full-body and the ball are physically simulated. Since our character does not have fingers yet, we attach the ball to the palm when the ball is in the hand and release the ball when it bounces off the ground. At the release, the velocity of the ball matches the velocity of the palm.

The motion generator was learned from a basketball data set of 12 minutes, which includes a variety of motor skills, such as walking, running, rapid turning and hand switching while dribbling the ball continuously. Since we manually labelled the timing of catching and releasing the ball in the training data, the motion generator outputs contact flags that indicate when to release the ball and when to catch the ball with which hand. We also incorporate the ball bouncing

reward suggested by Lee et al. [2018b] into the motion generator such that it can produce the prediction of the ball trajectory.

The control policy is learned in two steps. At the first step, we train the dribbling motion without the ball. The second step of learning refines the arm and hand motions to push the ball and catch it precisely. The future prediction from the motion generator continuously informs the release and catch timing and the ball position. The reward of the second step includes two additional terms r_{ball} and r_{hand}.

$$\begin{aligned} r_{\text{ball}} &= \exp\Big(-\frac{1}{\sigma_b^2} \parallel \hat{b} - b \parallel^2 \Big), \\ r_{\text{hand}} &= \exp\Big(-\frac{1}{\sigma_h^2} \parallel v_{\text{hand}} - v_{\text{ball}} \parallel^2 \Big), \end{aligned} \tag{18}$$

where b and $\hat{b}$ are the simulated and predicted positions of the ball, respectively. r_{ball} rewards a good match of the simulated and predicted trajectories of the ball. r_{hand} penalizes the mismatch of the hand velocity and the ball velocity at the release moment. Putting the terms into the reward function in Equation(11),

$$r = r_q \cdot r_v \cdot r_{\text{e}} \cdot r_{\text{com}} \cdot r_{\text{ball}} + w_{\text{hand}} r_{\text{hand}}. \tag{19}$$

We added r_{hand} with its weight instead of multiplying it to the other terms because the hand-ball velocity matching is a sparse term that is rewarded only at the release moment.

6.4 Ablations

We evaluated the effectiveness of data-driven prediction and adaptive sampling for policy learning from large data sets (see Figure 6). If the two technical components are ablated, our tracking controller is reduced to DeepMimic by Peng et al. [2018], which serves as a baseline of the comparison. We compare three algorithms: The baseline algorithm (Algorithm 1), the baseline algorithm driven by prediction (Algorithm 2), and the baseline algorithm with both prediction and adaptive sampling (Algorithm 3).

The test data were created from a basketball data set of 12 minutes playtime, which generated a larger data set (5 hours) through data augmentation. We randomly selected four data sets of 200 frames, 800 frames, 3,200 frames, and 12,800 frames. We compared the completion rate of the three algorithms for four data sets. If the algorithm is rewarded perfectly at all frames, the completion rate is 1.0. Otherwise, the completion rate is smaller than 1.0. Motion capture data are imperfect and have many small glitches. Some body segments may penetrate through the ground. The stand foot may stay in the air slightly above the ground. Shallow self-collisions and inter-penetrations are abundant even in high-quality motion capture data. Such small glitches are not noticeable if we do not zoom in the problematic area. Therefore, perfect tracking of the training data is often physically implausible. Fortunately, the DRL algorithm can gracefully deal with noises and glitches in the reference data. The simulation results are almost indistinguishable from the training data if the completion rate is larger than 0.8. The results look very satisfactory even at 0.4.

As expected, the baseline algorithm (Algorithm 1) works well with the smallest data set, but the performance does not scale well with bigger data sets. The discrimination power of a single phase variable is not sufficient to disambiguate future frames. Algorithm 2 replaces the phase variable with the future prediction from the motion generator. In our experiments, the prediction includes future frames at two temporal offsets 1/30 seconds and 10/30 seconds. The performance improves substantially for all data sets. Determining the number of future frames in the prediction is related to the diversity of actions in the training data. The single-frame prediction is sufficient with a small data set. The future prediction should disambiguate the possibilities in the near future, which may affect the dynamics of the character's body.

The full version of the algorithm with both data-driven prediction and adaptive sampling achieves even better performance of tracking for all four data sets. One might notice the undulation of the learning curves for Algorithm 3. The curve seems to reach its plateau and then undulate repeatedly. This phenomenon is related to the glitches in the motion data. Adaptive sampling puts a lot of samples in the problematic area with low average cumulative rewards. Those samples are wasted because there remains no room for further improvements. So, it is reasonable to terminate the learning process before such undulations occur.

The ablation study shows that the data-driven modeling is an essential step for physics-based simulation. The prediction from the motion generator not only improves the learning of control policies, but makes it possible to control physics-based characters interactively.

Table 2. Resilience tests with and without physical response learning. We measured how many impulses the pedestrian character can withstand while maintaining its balance. The balls are thrown towards the character at 9 meter/second velocity.

Ball weight(kg)	Without learning	With learning
4.0	79.2 %	98.3 %
4.5	59.8 %	85.5 %
5.0	23.7 %	55.9 %
5.5	4.0 %	37.7 %
6.0	1.3 %	15.4 %

6.5 Learning Physical Responses

No physical hitting, pushing, or physical interactions were captured in our motion data. The actors only pretended to push, punch, and kick in the motion capture sessions. Our motion generator never learned how the character might respond to physical interactions and perturbations. The lack of physical interactions in the training data makes the simulated character stiff, less responsive, and less resilient in the physics simulation.

To address this issue, we collected the simulated responses against external impulses and re-trained the motion generator with the simulated data. The pedestrian character was originally trained with 20,000 frames of the augmented trajectory. We additionally collected 3,000 frames of simulated data, in which the character was hit 30 times by 3.8 kg balls thrown at 9 meter/second velocity. The weight and velocity of the ball were determined so that the character can withstand the impulses. It turns out that the motion generator re-trained with the mixed data of 23,000 frames improves

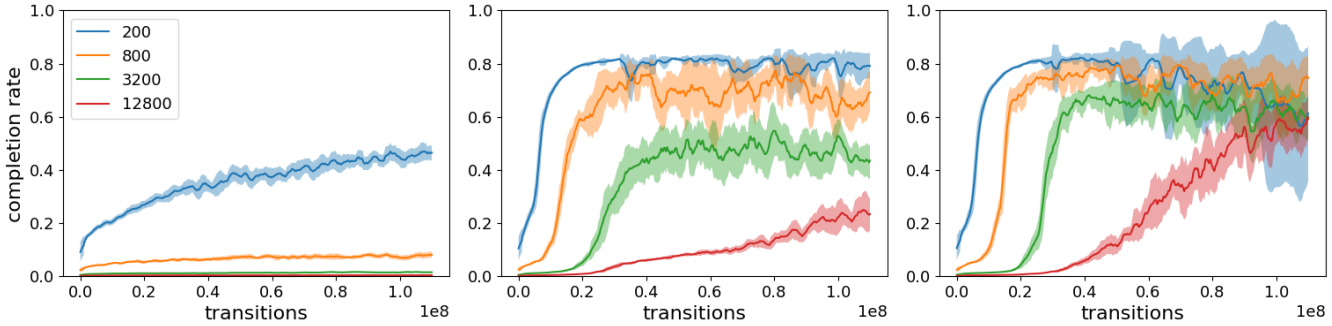

Fig. 6. The ablation study for tracking control. The x-axis is the number of transitions the algorithm sampled. The y-axis is the completion rate. (Left) The baseline algorithm. (Middle) The baseline algorithm driven by data-driven prediction. (Right) Our algorithm with both data-driven prediction and adaptive sampling.

the robustness and resilience of the simulated character significantly (see Table 2).

7 DISCUSSION

We presented a predict-and-simulate framework for creating interactive physically-simulated characters from unorganized motion data, which include a range of challenging motor skills. The kinematic aspects of the motion data are learned in an encoder-recurrent-decoder network, while the dynamic aspects of the motor skills are learned in a single policy network. This network architecture combines the interactivity of motion data modeling and the responsiveness of physics-based control to create simulated characters that can physically interact with each other and with the environment.

Although our experiments demonstrated the flexibility of this approach, there are still numerous limitations to be addressed in future work. Our characters look stiff at the presence of unexpected perturbations. It is a common limitation of example-guided tracking approaches, which seek for a trade-off among tracking accuracy, energy efficiency, and motion stiffness. Improving tracking precision requires higher gains and stiffer control dynamics. The use of PD servos as an intermediary in torque generation is another reason for stiff dynamics systems because it is non-trivial to incorporate an energy minimization framework into PD controllers. Torque-driven learning without PD servos might be a favorable direction to pursue for improving energy efficiency and alleviating motion stiffness. Achieving agile, yet compliant control policies is a challenging research goal.

Simulating physical interactions, such as hit and reaction, requires the understanding of force, impact, and pressure at the contact points. Since our motion data lack such an interaction context, simulated punches and kicks often lack sufficient power. In fact, the character just pretends to punch and kick while tracking the reference trajectory without proper interaction contexts. It would be interesting to capture, learn, and synthesize the physical interactions between multiple agents [Kry and Pai 2006].

The learning process is computationally demanding, often requiring several days per data set. The 20-minute basketball data set was the largest in our experiments. Learning a policy from the data set took 96 hours. It seems the learning time is highly correlated with the diversity and difficulty-level of motor skills, but the size of data augmentation is less relevant.

We believe this work nevertheless is the first step towards frameworks in which interactive characters learn a broad range of challenging motor skills, a variety of physical interactions, habitual patterns, social customs, sports rules, and dance choreography automatically from truly large data sets.

ACKNOWLEDGMENTS

This work was supported by Samsung Research Funding Center under Project Number SRFC-IT1801-01.

REFERENCES

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

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 (2013), 1405–1414.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. 2017. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748* (2017).

Jernej Barbič, Marco da Silva, and Jovan Popović. 2009. Deformable Object Animation Using Reduced Optimal Control. *ACM Trans. Graph.* 28, 3, Article 53 (2009).

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

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. ACM, 1.

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*. 4346–4354.

Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. 2017. Learning human motion models for long-term predictions. In *2017 International Conference on 3D Vision*. 458–466.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based Inverse Kinematics. *ACM Trans. Graph.* 23, 3 (2004), 522–531.

Sehoon Ha and C. Karen Liu. 2014. Iterative Training of Dynamic Skills Inspired by Human Coaching Techniques. *ACM Trans. Graph.* 34, 1, Article 1 (2014).

Sehoon Ha, Yuting Ye, and C. Karen Liu. 2012. Falling and Landing Motion Control for Character Animation. *ACM Trans. Graph.* 31, 6, Article 155 (2012).

Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. 2014. Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 51.

Perttu Hämäläinen, Joose Rajamäki, and C Karen Liu. 2015. Online control of simulated humanoids using particle belief propagation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 81.

Félix G Harvey and Christopher Pal. 2018. Recurrent transition networks for character locomotion. *arXiv preprint arXiv:1810.02363* (2018).

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

Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*. 4565–4573.

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

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 (2016).

Kyunglyul Hyun, Kyungho Lee, and Jehee Lee. 2016. Motion grammars for character animation. In *Computer Graphics Forum*, Vol. 35. 103–113.

Eunjung Ju, Jungdam Won, Jehee Lee, Byungkuk Choi, Junyong Noh, and Min Gyu Choi. 2013. Data-driven Control of Flapping Flight. *ACM Trans. Graph.* 32, 5, Article 151 (2013).

Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. 2009. Synchronized Multi-character Motion Editing. *ACM Trans. Graph.* 28, 3, Article 79 (2009).

Paul G. Kry and Dinesh K. Pai. 2006. Interaction Capture and Synthesis. *ACM Trans. Graph.* 25, 3 (2006), 872–880.

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*. Eurographics Association, 129–138.

Taesoo Kwon and Jessica K. Hodgins. 2017. Momentum-Mapped Inverted Pendulum Models for Controlling Dynamic Human Motions. *ACM Trans. Graph.* 36, 4, Article 145 (2017).

Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. *ACM Trans. Graph.* 21, 3 (2002), 491–500.

Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. 2018a. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software* 3, 22 (2018), 500.

Jehee Lee and Kang Hoon Lee. 2004. Precomputing avatar behavior from human motion data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 79–87.

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018b. Interactive Character Animation by Learning Multi-objective Control. *ACM Trans. Graph.* 37, 6, Article 180 (2018).

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

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

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010b. Motion Fields for Interactive Character Locomotion. *ACM Trans. Graph.* 29, 6, Article 138 (2010).

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

Libin Liu and Jessica Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Trans. Graph.* 36, 4, Article 42a (2017).

Libin Liu and Jessica Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 142 (2018).

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 (2016).

Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. 2012. Terrain Runner: Control, Parameterization, Composition, and Planning for Highly Dynamic Motions. *ACM Trans. Graph.* 31, 6, Article 154 (2012).

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. 2018. Neural probabilistic motor primitives for humanoid control. *arXiv preprint arXiv:1811.11711* (2018).

Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. 2017. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201* (2017).

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Trans. Graph.* 31, 6, Article 153 (2012).

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.* 31, 4, Article 43 (2012).

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

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.* 36, 4, Article 41 (2017).

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

Hyun Joon Shin and Jehee Lee. 2006. Motion synthesis and editing in low-dimensional spaces. *Computer Animation and Virtual Worlds* 17, 3-4 (2006), 219–227.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph.* 26, 3, Article 107 (2007).

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

Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

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

Jie Tan, Karen Liu, and Greg Turk. 2011. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications* 31, 4 (2011), 34–44.

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*. IEEE, 4906–4913.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. 2014. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1168–1175.

Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-optimal character animation with continuous control. *ACM Trans. Graph* 26, 3, Article 7 (2007).

Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B Cheng, Jehee Lee, and Tong-Yee Lee. 2010. Real-time physics-based 3d biped character animation using an inverted pendulum model. *IEEE transactions on visualization and computer graphics* 16, 2 (2010), 325–337.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2010. Optimizing Walking Controllers for Uncertain Inputs and Environments. *ACM Trans. Graph* 29, 4, Article 73 (2010).

Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. 2012. Optimizing Locomotion Controllers Using Biologically-based Actuators and Objectives. *ACM Trans. Graph.* 31, 4, Article 25 (2012).

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. 2017. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*. 5320–5329.

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

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

Yuting Ye and C. Karen Liu. 2010. Optimal Feedback Control for Character Animation Using an Abstract Model. *ACM Trans. Graph.* 29, 4, Article 74 (2010).

KangKang Yin, Kevin Loken, and Michiel Van de Panne. 2007. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.* 26, 3, Article 105 (2007).

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 (2018).

Yi Zhou, Zimo Li, Shuangjiu Xiao, Chong He, Zeng Huang, and Hao Li. 2018. Auto-conditioned recurrent networks for extended complex human motion synthesis. *arXiv preprint arXiv:1707.05363* (2018).

Victor Zordan and Jessica K Hodgins. 2002. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 89–96.

Victor Zordan, Adriano Macchietto, Jose Medina, Marc Soriano, and Chun-Chih Wu. 2007. Interactive dynamic response for games. In *Proceedings of the 2007 ACM SIGGRAPH symposium on Video games*. 9–14.