UniCon: Universal Neural Controller For Physics-based Character Motion

TINGWU WANG, NVIDIA, University of Toronto, Vector Institute, Canada
YUNRONG GUO, NVIDIA, Canada
MARIA SHUGRINA, NVIDIA, University of Toronto, Vector Institute, Canada
SANJA FIDLER, NVIDIA, University of Toronto, Vector Institute, Canada

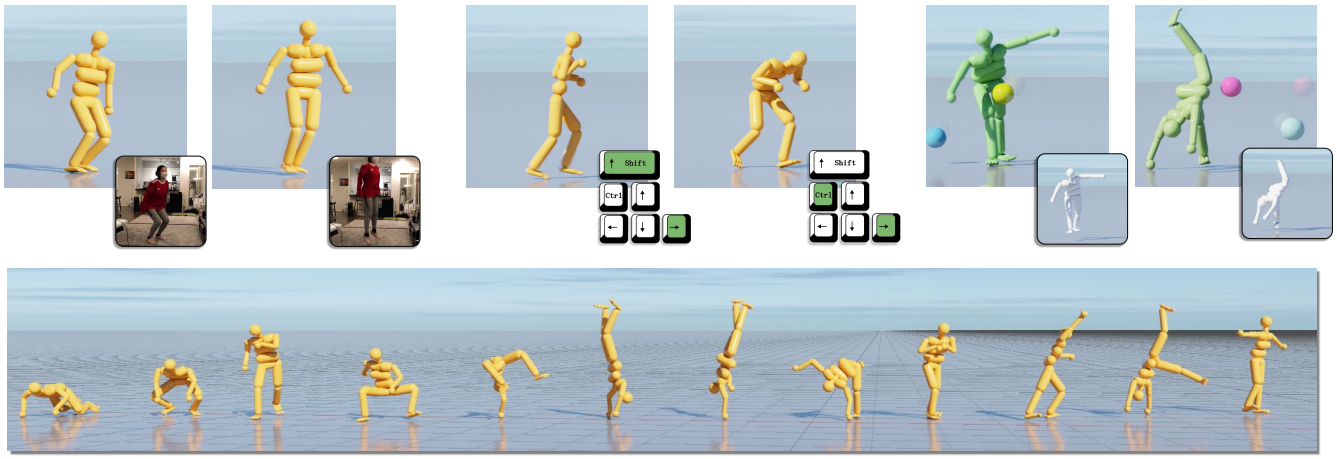

Fig. 1. UniCon provides physically valid control for thousands of diverse motions with a single trained model and can accept a variety of interactive inputs. The top figures show interactive control examples using video stream, keyboard input and reference motions with environment perturbations. The bottom figure shows UniCon's ability to generate smooth transitions and adapt on-the-fly to interactive motion stitching. Project page is https://nv-tlabs.github.io/unicon/.

The field of physics-based animation is gaining importance due to the increasing demand for realism in video games and films, and has recently seen wide adoption of data-driven techniques, such as deep reinforcement learning (RL), which learn control from (human) demonstrations. While RL has shown impressive results at reproducing individual motions and interactive locomotion, existing methods are limited in their ability to generalize to new motions and their ability to compose a complex motion sequence interactively. In this paper, we propose a physics-based universal neural controller (UniCon) that learns to master thousands of motions with different styles by learning on large-scale motion datasets. UniCon is a two-level framework that consists of a high-level motion scheduler and an RL-powered low-level motion executor, which is our key innovation. By systematically analyzing existing multi-motion RL frameworks, we introduce a novel objective function and training techniques which make a significant leap in performance. Once trained, our motion executor can be combined with different high-level schedulers without the need for retraining, enabling a variety of real-time interactive applications. We show that UniCon can support keyboard-driven control, compose motion sequences drawn from a large pool of locomotion and acrobatics skills and teleport a person captured on video to a physics-based virtual avatar. Numerical and qualitative results demonstrate a significant improvement in efficiency, robustness and generalizability of UniCon over prior state-of-the-art, showcasing transferability to unseen motions, unseen humanoid models and unseen perturbation.

CCS Concepts: • **Computing methodologies** → **Animation**; *Physical simulation*; *Control methods*; *Reinforcement learning*.

Additional Key Words and Phrases: physics-based character animation, reinforcement learning, interactive control, real-time graphics

1 INTRODUCTION

Physics-based animation can provide more realistic motions and richer interactions with the environment compared to traditional keyframe-based animation methods. However, film and video game industries have not yet adopted physics-based animation since the available controllers are still very limited in the number of supported motions, animation quality, interactiveness and efficiency.

The field of physics-based animation has recently seen a plethora of data-driven techniques using deep reinforcement learning (RL). Data-driven techniques promise scalability to a wide variety of motions by learning directly from human demonstrations. Most of the RL physics-based animation algorithms fall under the umbrella of imitation learning [??????], where the reward signals are given based on the distance or similarity between the generated and the target motions. A policy network, which maps the current state of the character to the torques applied to the joints, is then optimized to minimize this distance. Policies trained with imitation learning have shown success in driving a virtual character to naturally follow the reference target motions in a physically plausible way.

However, existing physics-based RL controllers are extremely inefficient to train, requiring hours or days of training to reproduce even a single motion. Furthermore, most of the existing controllers

have shown little diversity in motions, and usually lack robustness to perturbations in the environment. To increase the range of supported motions and allow for better user interaction, new methods are proposed by, for example, combining the controller with a powerful reference motion generator [???], or utilizing a hierarchical control system [????]. However, these methods usually specialize towards certain applications and support limited number of motions. These controllers are typically trained and tested on the same motion dataset and have not demonstrated generalization to unseen motions.

In this paper, we propose a physics-based *universal* neural controller (UniCon) which greatly improves training efficiency, robustness, motion capacity and generalizability, allowing for a wide range of real-time interactive applications. UniCon consists of two components: 1) a low-level motion executor that generates physics-based control signal which drives the character to follow a target reference motion, and 2) a high-level motion scheduler which converts various high-level inputs (for example, keyboard commands) into a target reference motion. A powerful and robust motion executor is the key innovation of our work. We introduce several components that allow us to train our motion executor on large-scale motion datasets of diverse motion styles using reinforcement learning. In particular, we utilize a constrained multi-objective reward optimization, a motion balancer and a policy variance controller, which enable efficient and robust policy learning. Once the low-level motion executor is trained, UniCon can utilize different motion schedulers for real-time interactive applications. UniCon can be used to perform keyboard-driven control, compose user-specified motion sequences, and supports teleporting a person captured on video to a physics-based virtual avatar.

Our experiments demonstrate key improvements of UniCon over previous work:

- **Generalization:** UniCon can be used to imitate motions which are unseen during training. Natural transition skills between motions are learnt automatically without the need of recording specific training samples in the dataset.
- **Robustness:** UniCon produces robust control even when the source motion is of poor quality. UniCon demonstrates *zero-shot* robustness to environment obstacles that are not seen during training, such as projectiles. Our model can also adapt to characters with widely varying mass, or slower or faster motion than the motions seen during training.
- **Interactive Applications:** Generalization and robustness allow many modes of interactive control, ranging from keyboard commands, noisy pose tracking from video capture, and user-specified sequences of locomotion or acrobatic motions, without having to retrain or fine-tune separate models for each application.
- **Learning Efficiency:** Our learning algorithm has better sample efficiency and asymptotic performance compared to existing baselines.

2 RELATED WORK

The problem of generating plausible human motion on the fly based on user input or a target goal is a long standing problem. We first discuss methods of motion generation which do not adhere to the laws of physics in Section **??**, then cover physics-aware methods in both non-interactive and interactive settings in Section **??** and **??**. We do not discuss all methods in character animation and refer readers to [**?**] for a comprehensive overview.

2.1 Keyframe Based Animation

Keyframe-based animation systems or kinematic systems can be used to interactively animate virtual characters by exploiting pre-recorded motions or human authored data. While often enabling better interactive control compared to their physics-based counterparts, these systems result in motions that are not physically accurate and brittle to perturbations in the environment.

Motion graph [???], as a pioneer in interactive animation, connects motions from the dataset based on the character's states and user specifications. However, the discretized connection in motion graph can result in non-smooth transitions, and the generated motion is not very responsive to changes in direction or perturbations. Follow up works improve motion graph by introducing parameterization, planning or semantic analysis [????]. In motion field [**?**], the authors use reinforcement learning to choose the most appropriate next motion. Motion matching [**??**] on the other hand, utilizes unstructured motion data to search for the most appropriate future frames given the character's current states and user control input. Motion matching is considered by many as the state-of-the-art keyframe based animation technique due to its flexibility and ability to support a large variety of motions [**??**].

In [**??**], more focus is given to motion synthesis by learning a latent variable model. With advances in data-driven modeling via deep neural networks, powerful generative motion models, such as phase-functioned neural network (PFNN) [**?**] and auto-conditioned recurrent neural network [**?**] were proposed. PFNN trains a neural network on a labeled dataset, which predicts future character states based on the trajectory control signal and the character's current states. The idea can be extended to quadrupeds with different walking phases such as dogs [**?**]. Furthermore, data-driven auto-regressive method is capable of animating character-scene interactions with impressive quality [**?**]. In [**?**], the authors use a recurrent neural network to enable complex combinatorial basketball motion generation. In [**?**], an auto-regressive conditional variational auto-encoder is used to generate complex soccer motions. In [**?**], model-predictive control is applied to generate locomotion step plans. Motion retargeting across different skeletons can be obtained via data-driven training with skeleton-aware operators, as shown in [**?**].

While not physically plausible, keyframe-based algorithms are capable of producing a wide variety of complex motions, and can all be theoretically used as a high-level scheduler for our algorithm, which we showcase in our experiments.

2.2 Non-interactive Physics-based Methods

We first consider physics-based methods that aim to imitate a pre-recorded motion in a physically valid way without consideration of interactive control. One common approach is to formulate animation as an optimal control problem, or a reinforcement learning problem,

which aims to minimize the distance between the reproduced and the recorded motions.

Some of the early attempts include [?], where the authors reconstruct and animate a diverse set of captured motions with randomized sampling, which is mathematically similar to the random shooting algorithms in model-based reinforcement learning [???]. Other methods include [?], which uses model-predictive control to obtain natural walking motions by specifying a reward function for walking. These methods all assume knowledge of the forward dynamics and can therefore be categorized as model-based reinforcement learning algorithms. The performance can be further improved by designing better sampling schemes as shown in [??]. However, these model-based RL methods are usually time consuming to train. They are either not real-time during test time [?], or require several hours of training time per motion through an iterative planning process [?]. These methods are also more prone to fall into local minimas during complex motion planning and are less resistant to external perturbations, which makes them difficult to adapt to new environments.

In contrast, model-free reinforcement learning allows more capacity for turbulence and test-time efficiency. In generative adversarial imitation learning (GAIL) [??], a neural controller is optimized with generative adversarial training [?], where a discriminator is used to distinguish the target motion from the generated ones. The neural controller is trained iteratively to compete with the discriminator by generating motions that are closer to the target ones. In [?], GAIL is further extended with a variational auto-encoder, which encodes global motion information, such as style. Results demonstrate that the controller can reproduce walking motions with several different styles. In DeepMimic [?], the authors show that deep model-free reinforcement learning can be used to train animation controllers for a wide range of skills. Their observation function includes the current states of the humanoid agents and a phase variable to encode time information of the frame. A policy network generates control signal to minimize the distance between the resulting next frame and the corresponding target frame. In [?], the authors further show that DeepMimic can be used to reproduce motions captured by pose estimators. Similar to the original DeepMimic method, their method requires training with RL on the motion frame data and is not real-time. In [?], the idea is further extended such that the agent can use first-person perception as input observation. Model-predictive control is used to model realistic eye and head movements in [?].

The work most related to ours is [?], which feeds a set of future frames from a dataset into the network to produce control signal that tracks these future frames. Being one of the earlier attempts to obtain a multi-motion animation controller, this algorithm is not used for interactive control due to its performance limitations on motion imitation and lack of robustness. In our paper, we study and identify several key issues that prevent successful training of an animation system on large-scale motion datasets, and propose a technique to significantly improve performance.

Recently, several works on constructing complex task-driven long motion sequences using a hierarchical RL framework have been proposed [????]. These methods usually focus on solving one or two specific tasks, and the number of supporting motions required is usually quite limited.

2.3 Interactive Physics-based Methods

Interactive control is typically done via a keyboard or a gamepad input. Earlier attempts such as [??] aimed to solve bipedal locomotion for humanoids using a high-level footstep planner that takes control and terrain information as input. In recent work [?] (DReCon), complex interactive locomotion skills of the full humanoid are enabled, where motion matching is used as a high-level planner that sends future kinematics states to a proportional-derivative (PD) controller. A neural network is used to further generate corrective control on top of PD control targets generated by the motion matching system. This method produces impressive and realistic locomotion results and can be regarded as the state-of-the-art interactive physics-based method for locomotion. Similarly, the idea can also be extended to quadruped locomotion problems as shown in [?]. In [?], the authors demonstrate interactive control of several complex acrobatic skills by designing a high-level control graph that schedules the reasonable motion fragments. In [?], deep Q-learning was combined into the fragment scheduler to further improve the scheduling performance. The authors also extend the proposed method to handle complex basketball controls in [?]. However, as mentioned in their papers and later shown in our experiments, the above methods cannot generalize to unseen motions due to severe overfitting. The number of motions supported by these methods is also limited. Our method, on the other hand, can be applied to unseen motions and is capable of generalizing to a larger number of motions of drastically different styles.

Notable recent, concurrent work [?] also considers learning large number of motions. In [?], the authors first divide the motions into separate clusters, then use different networks for different clusters. UniCon instead focuses on transferability, zero-shot robustness, and exploits capacity by systematically rethinking the training framework and proposing new techniques.

Besides using the keyboard control to generate future states, interactiveness can also be established by directly encoding control information into the observation function, such as using a user-specified one-hot clip-selection vector [?]. In [?], a recurrent neural network is used to predict task-driven future frames. A motion or intention vector can be interactively controlled by the user. The amount of motions supported in this encoding scheme is usually quite limited. In our experiments, we show that the performance drops catastrophically with increasing number of motions and the learnt controller exhibits virtually no ability to transfer to new motions.

Our proposed UniCon can also be used for physics-based interactive video control, which has not been widely studied in existing research. In real-time interactive video control, we directly teleport a person captured by camera into a virtual physics-based avatar.

3 OVERVIEW

The UniCon framework consists of two levels: a **high level motion scheduler** which takes as input interactive high-level control such as keyboard command or video and generates target motion frames, and a **low-level motion executor** which produces physics-based animation based on the target motion frames. The low-level motion executor presents the key innovation of our work, and can be used in

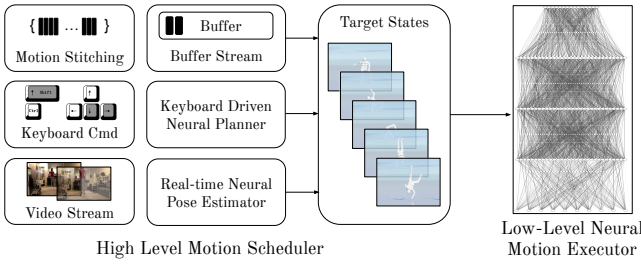

Fig. 2. Overview of UniCon. Our model consists of (**right**) an RL-powered low-level motion executor that is able to physically animate a given (non-physically plausible) sequence of target motion frames. The low-level motion executor can work in conjunction with a plethora of (**left**) high-level motion schedulers which produce target motion frames.

conjunction with a variety of different high-level motion schedulers. Figure **??** shows the overview of our model.

We first introduce the low-level executor in section **??**, and then separately discuss important design decisions that enable its robust training in a multi-task setting in section **??**. In section **??**, we describe how different high level motion schedulers can interact with our low-level executor, resulting in various interactive applications. Section **??** provides implementation details, including information about the humanoid model and physics engine used in our work. In section **??**, we showcase our method both qualitatively and quantitatively, and provide benchmarks against existing work.

4 LOW LEVEL MOTION EXECUTOR

The low-level motion executor assumes input in the form of target character states. These states come, for example, from mocap data, or can be produced by a high level motion scheduler. The low-level controller is implemented as a policy neural network that outputs a physics-based control signal, which drives the character to closely follow the target states.

We denote the state of the character as $\mathcal{X}$. To describe the state of a character, we consider the following information: the root position $p^r \in \mathcal{R}^3$, the root rotation quaternion $q^r \in \mathcal{R}^4$, the joint position $p^j \in \mathcal{R}^{3J}$, and the joint rotation quaternion $q^j \in \mathcal{R}^{4J}$ where J is the number of joints. We note that this information is redundant in that, for example, the joint position p^j can be inferred from p^r, p^j, q^r. We also consider the first order information from the four states, i.e., the root translation velocity $\dot{p}^r$, the root angular velocity $\dot{q}^r$, the joint translation velocity $\dot{p}^j$, and the joint angular velocity $\dot{q}^j$. The state thus takes the following form:

$$\mathcal{X} = \left[p^r, q^r, p^j, q^j, \dot{p}^r, \dot{q}^r, \dot{p}^j, \dot{q}^j\right]. \tag{1}$$

We denote the time steps in the physics engine as t. At time step t, the state of the character is denoted as $\tilde{\mathcal{X}}_t$. The use of $\mathcal{X}$ with or without tilde symbol ˜ is to differentiate between the actual state and target state. The target states, which are input to the low-level controller, are denoted as $[\mathcal{X}_{t+1}, ..., \mathcal{X}_{t+\tau}]$, where τ is the length of the target frames. In section **??**, we describe how to generate the target states from training datasets, or from interactive high-level schedulers that take control signal c_t. We here focus on our low-level motion executor.

4.1 Observation Function

We first introduce our observation function s_t, which encodes information about both the current state $\tilde{\mathcal{X}}_t$ and the target future states $[\mathcal{X}_{t+1}, ..., \mathcal{X}_{t+\tau}]$. In our RL formulation, the agent's controller takes input the observation function, and generate the corresponding control signal as output. We propose to use an agent-centric state encoding operator T_{p^r,q^r}, which transforms the quaternion, translation and the corresponding velocity with respect to the root p^r, q^r. The agent-centric local state observation function $o(\mathcal{X})$ can be written as:

$$o(\mathcal{X}) = \left[p^r(z), T_{p^r,q^r}\left(q^r, p^j, q^j, \dot{p}^r, \dot{q}^r, \dot{p}^j, \dot{q}^j\right)\right], \tag{2}$$

where $p^r(z)$ here indicates that we only use the z value out of all the xyz coordinate values of p^r. We note that in equation **??**, we transform the target state $\mathcal{X}$ information with respect to the root of each target state $\mathcal{X}$, rather than with respect to the root of the current actual state $\tilde{\mathcal{X}}$. Empirically, the two choices do not make noticeable performance difference. By processing the character's actual state in an agent-centric way, we increase the generalization of the observation function. The second part of the observation function is a relative information function $y_{\tilde{\mathcal{X}}}(\mathcal{X})$ that extracts the relative root coordinate offset between future target states and the current actual state, i.e.,

$$y_{\tilde{\mathcal{X}}}(\mathcal{X}) = \left[T_{\tilde{p}^r,\tilde{q}^r}\left(p^r, q^r\right)\right]. \tag{3}$$

By combining both the local state information $o(\mathcal{X})$ and the relative information between states, the observation vector s_t takes the following form:

$$s_t = \left[o(\tilde{\mathcal{X}}), o(\mathcal{X}_{t+1}), ..., o(\mathcal{X}_{t+\tau}), \quad y_{\tilde{\mathcal{X}}}(\mathcal{X}_{t+1}), ..., y_{\tilde{\mathcal{X}}}(\mathcal{X}_{t+\tau})\right]. \tag{4}$$

We avoid the use of absolute information in world space, which helps the agent to learn generalized features for control.

4.2 Controller

The choice of a controller is an important one. In existing RL-based animation algorithms, the proportional-derivative (PD) controller has been commonly used instead of the alternative torque-based controller that decides how much force to apply on each joint. We here argue in favour of torque-based controllers for our setting. In particular, in the experimental section we show empirically that, while PD controller usually has good sample efficiency and performance in training, it also tends to severely overfit. Although we do not present a mathematical explanation for this phenomena, a general rule of thumb we consider here is to avoid providing the controller with additional observation features, pre-processing or post-processing. In [?], the authors show that the Atari RL agents actually strongly focus on the scoreboard or the timer in their policies instead of paying attention to the actual game screen, suggesting the risk of RL agents using unexpected features from observation function. We also refer to [?] for comparison study.

Therefore, we use a torque based controller for UniCon, which we denote as $\pi(a_t|s_t)$, where a_t is the concatenation of torques applied to each joint. We use a fully-connected neural network (Multilayer Perceptron, MLP), and denote the network weights as θ_π. Since our controller is tasked to master a much larger number of

skills compared to [??], we also use a much larger neural network. Specifically, in this paper we use three hidden layers of 1024 units, and ablate this choice in section ??.

4.3 Constrained Multiobjective Reward Optimization

Similar to [??], we define the reward as a sum of several terms that measure the difference of the target state and the actual state on various statistics, i.e.,

$$r(s_t) = r_t(\tilde{\mathcal{X}}, \mathcal{X}) = w_{p^r} r_{p^r} + w_{q^r} r_{q^r} + w_{p^j} r_{p^j} + w_{q^j} r_{q^j} + w_{\dot{q}^j} r_{\dot{q}^j}, \quad (5)$$

where the weight coefficients $(w_{p^r}, w_{q^r}, w_{p^j}, w_{q^j}, w_{\dot{q}^j})$ are respectively $(0.2, 0.2, 0.1, 0.4, 0.1)$, which were empirical numbers first used in DeepMimic [?]. We use the exact same reward function r_{q^j} for joint quaternion deviation, $r_{\dot{q}^j}$ for joint angular velocity deviation, and root position deviation r_{p^r} (centor-of-mass), as the ones in DeepMimic [?]. However instead of penalizing only the mismatch of positions for end-effectors such as hands and feet, we penalize all joints, as accurate representations of many motions require attention to all joints rather than hands and feet only. We also separately penalize the root rotation similar to how we penalize joint rotations as: $r_{q^r} = \exp(-2||\tilde{q^r} - q^r||^2)$.

It is common practice in RL-based animation algorithms to optimize the following objective: $J(\theta_\pi) = \mathbb{E}_{s_t \sim \rho_{\theta_\pi}(s)}(r(s_t))$. However, we show that directly optimizing the sum of the reward is problematic. This objective is essentially a multi-objective reward function, and mathematically every separate reward term is competing against the others in training. The competing effect is not apparent when training on very few motions, yet it is almost guaranteed to degenerate our learnt controller when the reward is dominated by certain reward terms [??]. A common resulting suboptimal behavior is the *moon-walk* behavior, where the agents get high reward on matching joint and root rotations, but almost completely ignore the reward for matching root position. Increasing the reward weight for matching the root position, however, would cripple the joint matching score and lead to "blurred" motions.

Therefore we propose the following constrained optimization objective:

$$\begin{aligned} \max_{\theta_\pi} J(\theta_\pi) =& \mathbb{E}_{s_t \sim \rho_{\theta_\pi}(s)}(r(s_t)), \\ & s.t.\ \ r_i(s_t) > \alpha_i,\ \forall r_i \in \left[r_{p^r}, r_{q^r}, r_{p^j}, r_{q^j}, r_{\dot{q}^j}\right], \end{aligned} \quad (6)$$

where α_i is the tolerance coefficient ensuring that no reward term is dominated by other reward terms.

For a practical algorithm, it is impossible to directly optimize equation ?? using existing RL algorithms off-the-shelf. However, we can maintain a soft version of the constraint $r_i(s_t) > \alpha_i$ by enforcing early termination separately for each term, i.e., we terminate the episode if any reward term drops below the tolerance threshold. Empirically, we find that $\alpha_i = 0.1\ \forall\ i$ works well.

5 TRAINING

We use proximal policy optimization (PPO) [?] to optimize UniCon. PPO has been commonly used in recent physics-based research including [??]. We refer readers to the original paper for the detailed PPO algorithm [?]. We here use the following surrogate objective denoted as L_{PPO} from PPO[1]:

$$\max_{\theta_\pi} \mathbb{E}\left[\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)} A_t - \beta \cdot \mathrm{KL}\left[\pi(\cdot|s_t)||\pi_{old}(\cdot|s_t)\right]\right], \quad (7)$$

where A_t is the estimated advantage function, π_{old} represents the policy weights which are fixed during the update, and β is the weight for the Kullback-Leibler divergence penalty that discourages over-confident updates, as proposed in [?]. The iterative update in equation ?? is sample-based, and we describe how we sample the motion and initial state in section ?? and ??. During training, we simultaneously use 4096 workers to generate training samples. The number of samples per iteration per worker is 64. We also point out that other potentially more powerful RL algorithms can be applied as well, such as for example [??].

5.1 Motion Balancer

As later described in detail in section ??, the training motion dataset contains classes with an imbalanced number of samples. Random sampling during training can lead to a policy that is dominated by one specific skill, for example walking, which makes for around 35.4% of the dataset. It is difficult to control the balance of the dataset if, for example, we wish to extend UniCon to train on a dataset generated from vastly available Youtube videos in the future. We do not want to discard unbalanced data samples which may contain useful information that can improve generalization to a broader set of skills. Furthermore, the class labels for motions are usually a mixture of rough and fine-grained labels, posing additional challenges in maintaining the balance of training samples.

We thus propose to use a *motion balancer*. In particular, we first build a hierarchical tree structure for class labels, similar to what was done in ImageNet [?]. For every motion, starting from class root node, we label their high-level class (major style such as walking), and then move downwards in the tree structure to the low-level classes (minor style such as forward or backward for walking). We do not limit the depth of this class hierarchy, such that for complicated motion we are able to generate more fine-grained class labels. For example, we can label a zombie-walking motion as `root-walking-forward-specialStyle-zombie`.

During training, we sample the motion by going down the hierarchical tree and uniformly sampling from all sub-node (child node) classes of the current node. We denote each node as v and its sub-nodes or child-nodes as $C(v)$. The sampling process can be described as

$$P(v_i|v) = \frac{1}{|C(v)|},\ \forall v_i \in C(v). \quad (8)$$

We note that the sampling probability for each motion can be calculated off-line once for the entire dataset. We also point out that motion balancer can be extended with other adaptive sampling schemes in the future. Later in section ??, we explain the collection of hierarchical dataset by using human-labor and existing pose estimators. In the experimental section ?? we show that with the motion balancer our controller is able to obtain significantly better performance for a variety of tasks.

[1]PPO is commonly referred to as a Policy Gradient (PG) method in current research. While PPO shares a lot of similarities with the original PG [?], it is considered as a trust region method by its authors and optimizes a slightly different loss function [?].

5.2 Reactive State Initialization Scheme

In DeepMimic, reference state initialization (RSI) was proposed to sample the state of a particular frame as the initial state. In our work, we further propose *reactive state initialization scheme* (RSIS), where the agent is initialized with a state of the frame k time-steps away from the actual target frame it is supposed to track. RSIS also includes a much larger noise added to the velocity and translation of the initialized stateas mentioned in later experiment sections. With RSIS, the agent learns how to self-adjust and catch up with the target states. In UniCon, we show that recovering skills can be learnt automatically without the need to train a separate recovery network as in [**?**], or adding recovering motions in the dataset. When used with a motion stitching scheduler our agent naturally transitions between stitched motions that may have large discontinuities. The robustness of our agent is also greatly increased, demonstrating extreme recovering skills from perturbations as shown in the accompanied videos. Empirically, we choose a frame offset with 5-10 time-steps, which does not lead to early termination of episodes while encouraging the character to learn recovering skills.

5.3 Policy Variance Controller

Another important consideration in training is avoiding bad local minima. Specifically, we consider the variance of the stochastic policy network $\pi(a_t|s_t)$. In PPO, a trainable vector $\hat{\sigma} \in \mathbb{R}^{|a_t|}$ is used to represent the diagonal Gaussian policy standard deviation. Typically for single-motion or few-motion training, $\hat{\sigma}$ can be automatically learnt by optimizing equation **??**. However, with a large-scale dataset the variance is much more fragile during training. There is a design dilemma for the initial value. Initializing with a large variance can make the training hard to converge and thus generates divergent behaviors. Initializing with a small variance can lead to premature convergence and thus hurt the performance. For UniCon, we are inspired by [**?**] and use a similar exponential annealing on the policy's variance. We also notice that for a character different joints require variance of different scales. For example, the control variance of the toes should be intuitively and empirically smaller than the ones of the legs. We want to preserve the learnt differences between joints and therefore propose an adaptive variance update scheme as follows:

$$\begin{aligned} \hat{\sigma}' &\leftarrow \hat{\sigma} - \alpha_{lr} \nabla_{\hat{\sigma}} L_{PPO}, \\ \hat{\sigma}' &\leftarrow \begin{cases} \mathcal{Z}(\hat{\sigma}'), & l < \mathcal{L} \\ \hat{\sigma}', & \text{Else} \end{cases} \end{aligned} \tag{9}$$

where L_{PPO} is the loss defined in equation **??**, α_{lr} is the learning rate, and l is the current PPO training iteration. We define a control iteration range from 0 to $\mathcal{L}$, during which we linearly anneal the target average of $\log(\hat{\sigma})$ from the hyperparameters logstd_0 and logstd_F. $\mathcal{Z}$ is the operation where we linearly increase or decrease the log of each component of $\hat{\sigma}$ by the same amount, so that the average value matches the linear annealed target value. By doing this we preserve the learnt variance structure, which we show in section **??**.

6 HIGH LEVEL MOTION SCHEDULER

The high-level motion scheduler in our framework outputs target reference states from interactive control signal or from a motion dataset. We here discuss several different high level motion schedulers, and note that other existing schedulers can work in conjunction with our low-level executor. We denote the high-level motion scheduler as ϕ.

Dataset	Split	Num of Motions	Num of Frames	Avg Length
All	Train	2758	752648	272.8
	Test	594	187130	315
RunJumpWalk	Train	1376	290210	210.9
	Test	238	71262	299.4
Casual	Train	590	176698	299.5
	Test	108	42934	397.5
SportDance	Train	576	148182	257.2
	Test	66	35314	535.1
Animal	Train	244	109396	448.3
	Test	62	25954	418.6
Walk	Train	180	23322	129.5
	Test	48	4812	100.2

Table 1. In this table we show the statistics of each sub dataset we created.

The high-level motion scheduler outputs states of τ future frames as

$$[\mathcal{X}_{t+1}, ..., \mathcal{X}_{t+\tau}] = \phi_\theta \left(\{c_i\}_{i=t-\tau_c}^{t}, \{\tilde{\mathcal{X}}_i\}_{i=t-\tau_x}^{t}, t \right), \tag{10}$$

where $c_i, \tilde{\mathcal{X}}_i$ represent the control signals and actual robot state, with τ_c and τ_x representing respectively the history length to consider. θ in equation **??** represents the parameters or configurations of the scheduler. The planning length τ needs to be carefully chosen: it cannot be too long since this can make the executor hard to train and is more prone to overfitting. For real-time applications such as animation from video, it is also challenging to generate frames for the unforeseen future. We therefore choose a relatively short output length τ, with a value of 1 or 2.

By unifying input observation to our low-level motion executor in UniCon, we establish a universal and transferable framework for different sources of control input. In experiments, we show that the low-level executor trained with a large-scale motion dataset can be used directly with a scheduler for specific applications without the need of retraining.

6.1 Motion Dataset Training Scheduler

In this subsection, we discuss how one can utilize a motion dataset for the low-level executor. The motion dataset training scheduler is different from other schedulers which enable interactive applications, but serves as the backbone of UniCon, as it trains a powerful low-level executor to support other schedulers. During training we randomly sample a motion m and frame ID j and set this frame's state as the initial state for our agent, i.e. $\tilde{\mathcal{X}}_{t=0} = m(j)$. For time step i, our training scheduler outputs the state from motion m as

$$\phi_{\text{MocapData}}(t) = [m(t+j+1), m(t+j+2), ..., m(t+j+\tau)]. \tag{11}$$

The scheduler terminates and reschedules a new motion only when the motion has reached the end, or the target and actual states have deviated significantly, as discussed in section **??**.

6.1.1 Data Preparation. We utilize the widely available CMU Graphics Lab Motion Capture Database (CMU Mocap dataset) [?]. While motion dataset is commonly used in a physics-based system, the choice of train and test set splits has been generally overlooked. In our work, we carefully study the overfitting and transferability of the algorithms. We clean up the CMU Mocap dataset, and divide the data samples into training and testing sets based on several categories as presented in Table **??**. We create several smaller datasets by grouping via style and tasks. For example, `Casual` dataset contains casual motions such as sweeping the floor, cleaning dishes, and the majority of the motions are standing motions. `Animal` dataset contains motions where the actors imitate animals such as cats and dogs. The `All` dataset contains all remaining motions after filtering out infeasible motions or motions that are highly dependent on external objects such as stairs.

We split the dataset into a training set that has 80% of the frames and a test set that has the remaining 20%. One issue with the CMU Mocap dataset is that the motions are extremely unbalanced. We find that for all of the motions available, 35.4% of them are walking motions, and 25.5% of all motions are walking-forward motions. On the other hand, there are lots of classes with very few motion clips. Unbalanced data is a very common [?], which may bias the neural network and detriment performance. To build the training and test sets, we split motion classes individually, starting with least frequent categories. If there is a class with only one motion, we always place it in the test set. We then allocate large classes such as walking-forward to fill the remaining training and testing sets. By doing this, we make sure that the test set has a good variability of motions. While the training set is unbalanced across classes, our motion balancer discussed in section **??** helps in training our motion executor effectively. For UniCon we manually label the motions. While we used human labeled dataset in our experiments, we point out crawled unlabeled videos can be used to generate future datasets. Fine-grained hierarchical labels can be automatically generated with video action classifiers, which also produce hierarchical labels as shown in [**??**].

6.2 Video Stream Scheduler

Video can be used as an interactive control input for UniCon. Specifically, we consider a human who is captured by a camera and wants to teleport her/his motion onto a virtual avatar. We use a real-time pose estimator to estimate the 3D pose from video as shown in figure **??**. We use [?] in our work but any other 3D pose estimation approach can be utilized, such as [**??**]. Without the loss of generality, we assume that the pose estimator is parameterized by a convolutional neural network with weights θ_{CNN}, and we represent the pose estimator as $\mathcal{F}_{\theta_{CNN}}$. We denote the video frame at time step $t+1$ as $\mathcal{I}_{t+1}$. The estimator generates the following prediction:

$$\mathcal{P}_{t+1} = \left[p^r_{t+1}, q^r_{t+1}, p^j_{t+1}, q^j_{t+1}\right] = \mathcal{F}_{\theta_{CNN}}(\mathcal{I}_{t+1}) \tag{12}$$

A user datagram protocol (UDP) system connects the animation engine and the pose-estimator, sending $\mathcal{P}_{t+1}$ in a real-time fashion. Assuming we maintain a pose buffer of length τ_p, the first order information such as the linear and angular velocities are then interpolated from previous poses, i.e., $\left[\tilde{p}^r_{t+1}, \tilde{q}^r_{t+1}, \tilde{p}^j_{t+1}, \tilde{q}^j_{t+1}\right] =$ Interp$(\{\mathcal{P}^{t+1}_{i=t+1-\tau_p}\})$. We can write the video stream scheduler as

$$\mathcal{X}_{t+1} = \left[\phi_{\theta_{CNN}}(\mathcal{I}_{t+1}), \text{Interp}\left(\{\mathcal{P}^t_{i=t-\tau_i}\}\right)\right]. \tag{13}$$

Note that the accuracy of pose estimation approaches is not perfect. Our experiments show that our low-level executor can handle very noisy estimated poses with reasonable accuracy.

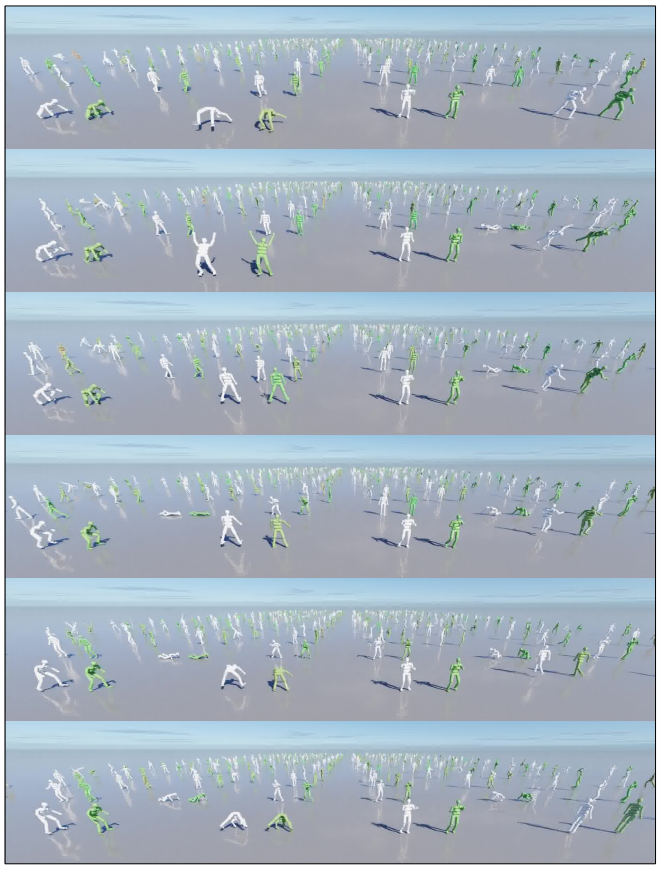

Fig. 3. The figures here visualize the training process, where the white characters are driven by ground-truth motions, and the green ones are trained characters. We use the motion dataset training scheduler to train our low-level motion executor. 4096 characters are used to generate training samples.

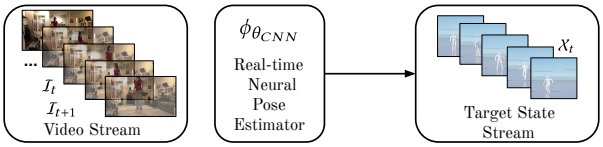

Fig. 4. Diagram of interactive control from video.

6.3 Keyboard Driven Interactive Control Scheduler

For Unicon, instead of training a hierachical interactive scheduler from scratch, we use Phase-functioned neural networks (PFNN) [?]

to process the keyboard commands and generate future states, as shown in figure **??**. We note that the future states generated by PFNN is not physics-based. In PFNN, one can control the walking direction of the agent, and choose the walking style from walking, jogging, crouching, etc. We refer readers to [**?**] for details regarding PFNN. We write the target state generation of PFNN as

$$\mathcal{X}_{t+1} = \phi_{\theta_{PFNN}}\left(\mathcal{X}_t, \{c_i\}_{t=t-\tau_c}^{t}\right). \tag{14}$$

Note that PFNN is an auto-regressive method where the previously generated state $\mathcal{X}_t$ is also an input for the generation of $\mathcal{X}_{t+1}$. Our experiments show that we do not require feeding the actual state $\tilde{\mathcal{X}}_t$ back into PFNN. Our low-level executor can automatically correct the accumulating mistakes in PFNN.

We also point out that besides PFNN, our algorithm can also be extended to use other keyframe based animation systems such as [**?**].

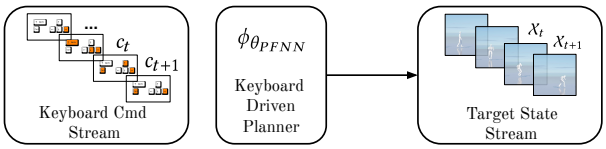

Fig. 5. Diagram of keyboard driven interactive control scheduler.

6.4 Motion Stitching Scheduler

Users can also interactively specify the motion for the character using a scheduler we call motion stitching scheduler. Motion stitching refers to directly stitching motions from database consecutively without worrying about proper transitions, as shown in figure **??**. We maintain a stitched target motion buffer $\mathcal{B}$. Before the current buffer finishes, one can interactively add another motion m with $|m|$ frames into the buffer, i. e.

$$\mathcal{B}.\text{push}(\{m(k)\}), \text{ where } k = 0, ..., |m| - 1. \tag{15}$$

We use spherical linear interpolation to add several transition target frames. At every time-step, motion stitching scheduler generates the next target state by popping a state like a FIFO buffer:

$$\mathcal{X}_{t+1} = \phi_{\mathcal{B}} = \mathcal{B}.\text{pop}() \tag{16}$$

We note that the stitching scheduler can be regarded as the simplest version of motion graph, where motion is animated one-by-one. However, we show that our controller can animate a wide variety of highly difficult acrobatic motions on-the-fly in a physically plausible way, some of which are unseen in training. Our character automatically makes smooth transitions between motions even though we do not have transition skills recorded in the training set. Empirical results suggest that by plugging in a more powerful interactive motion graph system, our controller can generate an even more diverse set of physically plausible motion sequences.

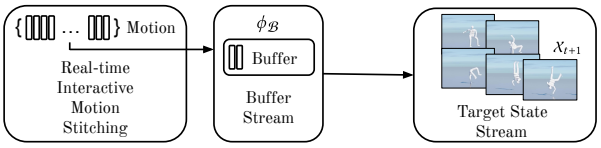

Fig. 6. Diagram of motion stitching scheduler.

7 ENVIRONMENT

7.1 Physics Engine

Our experiments are performed with a reinforcement learning simulator similar to OpenAI Gym [**?**]. The simulator is powered by the GPU-accelerated Flex physics engine as the core backend for physics simulation. We refer the reader to [**?**] for more implementation details of the simulator and [**?**] for the Flex physics engine solvers.

In our experiments, we use the CUDA-based Newton Preconditioned Conjugate Residual Method (PCR) solver for rigid-bodies provided by the Flex physics engine, with a simulation timestep of $1/60s$ and a simulation substep value of 4. We set the number of iterations taken by the solver per simulation step as 4, and the number of inner loop iterations taken by the solver per simulation step as 15. For scene simulation parameters, we set gravity to $9.8m/s^2$ downwards and use a value of 1.0 for coefficient of static and dynamic friction.

7.2 Humanoid Model

We design our humanoid model with the basic topology of a rigid-body representation modeled after the human body. Our humanoid includes 20 rigid-bodies and 35 degrees-of-freedom. Each degree-of-freedom is assigned an effort factor within the range of 50 to 600, to simulate the difference in strength of joints in the human body. This effort factor is taken into consideration when torque control is applied. The height and mass of the humanoid model resembles a realistic proportion of the human body, at $1.8m$ and $70kg$ respectively. The mass of each rigid-body in the humanoid model is proportionally distributed based on a rough estimate of the human body mass distribution. We use a fully symmetric humanoid model with respect to the left and right rigid-bodies and joints. We do not tune the parameters of the humanoid, and the learned policy generalizes across different models, which we refer to section **??**.

8 EXPERIMENTS

In this section we study the performance of the low-level motion executor (section **??**), the backbone of our algorithm, on our motion dataset. More specifically, we numerically compare our low-level executor with existing algorithms by modifying them to be trainable on a large-scale motion dataset. Results show that our low-level executor performs better in almost every metric (section **??**), and we analyze the key factors behind its success in section **??**. In addition, we demonstrate that, unlike prior work, our low-level executor exhibits zero-shot robustness to environment perturbations not seen during training (section **??**). These qualities of the low-level executor enable interactive applications outlined in section **??**. The

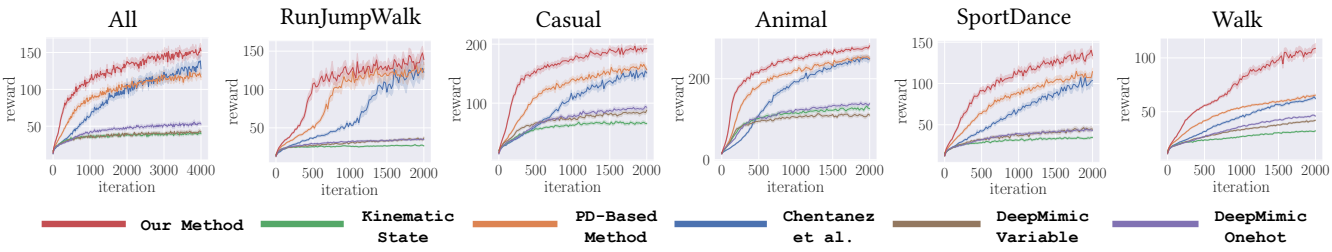

Fig. 7. The numerical training performance for our algorithm and baselines. We show the average sum of reward per episode. Our algorithm obtains better sample efficiency and performance.

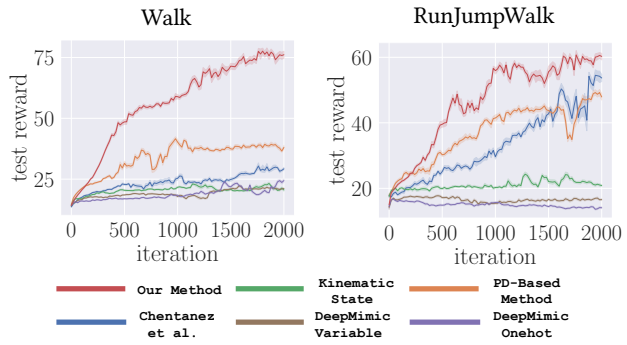

Fig. 8. Performance on the test set during training.

effectiveness of motion balancer and variance controller is shown with numerical ablation study. We also show visual ablation study on constrained multi-objective reward optimization and RSIS in the attached demo video.

8.1 Low-level Controller Baselines

We first introduce the baselines we use for comparisons here. We emphasize that constrained multi-objective reward optimization and the policy variance controller are crucial to training, without which, neither of our nor the baseline algorithms can be trained successfully. Therefore we apply these techniques equally to all baselines, and focus only on the controller design. We use the same network structure (1024x3) for every algorithm. We also tried the original structures specified in the papers, but the performance is worse than or similar to the ones with 1024x3.

PD-based Method: PD-based methods utilize a PD controller instead of a torque controller. During training, a neural policy network takes as input the current state and the target states from the dataset, and outputs the corrective offsets to the PD-control targets. This method is most similar to [**?**], except for the fact that [**?**] uses an online motion matching system to generate target states, whereas we directly use the target states from the dataset. We also use similar hyper-parameters from [**?**]. We show another simple variant of removing the actual state from the observation function and performing open-loop control, which we name as **Kinematic-State** baseline.

[?]: [**?**] is similar to PD-based methods, but the observation function of the policy network contains additional long-term information. A concatenation of future frames with $0, 4, 16, 64$ time-step offsets are fed as observations into the tracking network, which outputs PD targets to control the humanoid. We do not include a separate recovery agent as in the original paper, since it can be applied to any algorithm being compared here. We later show that a separate recovery network is not necessary with the existence of a powerful low-level controller.

DeepMimic-Onehot and DeepMimic-Variable: The original DeepMimic algorithm supports multi-motion training through the use of a one-hot vector to encode motion information into the observation function, which we name as **DeepMimic-Onehot**. Since the number of motions during training can be quite large, we also include another variant called **DeepMimic-Variable**, where we feed the ID (normalized between 0 and 1 for all motions in the train and test sets) of the motion to the observation function.

8.2 Training Performance

In figure **??**, we show the performance of our executor and the baseline methods on 6 datasets introduced in section **??**. Our executor consistently obtains better sample efficiency and performance across the datasets. Results with the PD-based method and [**?**] show that while PD-controllers are very good at reproducing single or few motions, they perform worse than torque-based controllers on large motion datasets. We note that the PD-based method, which only uses a small number of future frames in the observation functions, actually performs better than [**?**]. This could potentially be a cause of [**?**] feeding too much information into the network, some of which can contain information too far into the future that confuses the network and slows down the training process. We see poor performance from the Kinematic-State baseline, indicating that open-loop control is not enough for complex multi-motion animation tasks. We also see that both DeepMimic-Onehot and DeepMimic-Variable obtain poor performance, indicating their lack of model capacity.

8.2.1 Testing Performance. In this section, we study the generalization of algorithms to unseen motions. In figure **??**, we show the performance of our method and the baseline algorithms on the test set. The PD-based method and [**?**] reach their performance plateau very quickly on the test set, while training performance is still increasing. This over-fitting behaviour is most obvious in the two datasets shown in the figure. We suspect that this is due to the use

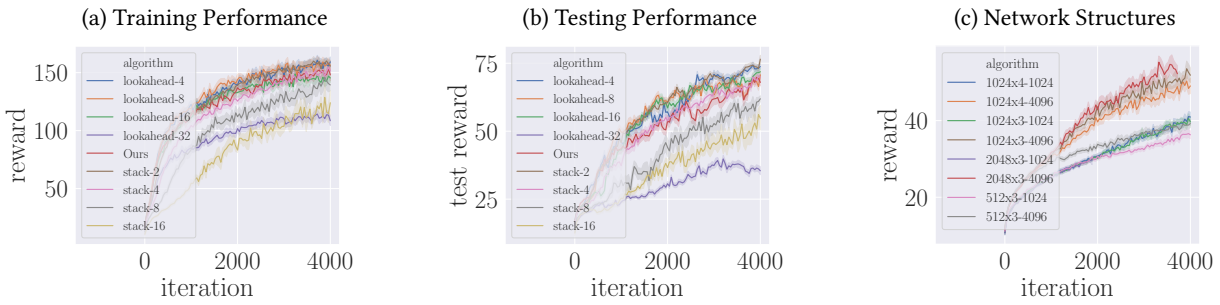

Fig. 9. Ablation study on model variants and network structures. In figure (c), we represent experiments in the format of "HiddenLayerSize"x"NumberOfLayers"-"NumberOfAgents". For example "1024x4-4096" represents an experiment on a network with 4 hidden layers of size 1024, trained with 4096 agents.

of PD-controllers. Over-fitting in deep reinforcement learning is still an under-explored topic, and we do not explore further into this direction.

On the other hand, the test performance of Kinematic-State, DeepMimic-Onehot and DeepMimic-Variable shows almost no improvement throughout the training process, demonstrating serious over-fitting as well.

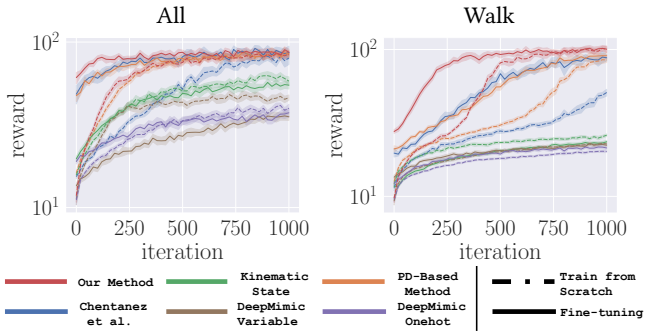

Fig. 10. The performance of transfer learning for different controllers, with and without a pretrained model.

8.3 Transfer Learning with Fine-tuning

Since the test set was not seen during training, for academic purposes only, we can reuse the test-set to study how the controllers' learnt features can be transferred to a dataset. In figure **??**, dashed lines represent models trained from scratch, and solid lines represent models with pre-trained knowledge. We show that our low-level controller is highly transferable, suggesting potential use cases where we can first train a model on an enormous dataset, then fine-tune the model on smaller datasets for better specialization. The PD-based method and [**?**] also have slightly worse but reasonable transferability, while Kinematic-State, DeepMimic-Onehot and DeepMimic-Variable show no or even negative transferability.

8.4 Ablation Study on Low-level Executor

In figure **??**, we study how the choice of target states and network structures affects performance of the network. We design two variants, the `lookahead` and `stack`. In `lookahead`-k variant, we use the

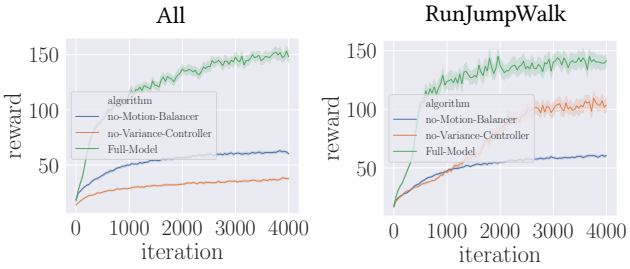

Fig. 11. This ablation study shows performance with and without variance controller and motion balancer.

target state from the kth future frame, instead of the next frame. In `stack`-k, we include all target states in k future frames into the observation function. As we can see from figure **??** (a), (b), the number of future frames from the high-level scheduler (i. e. τ), does not visibly affect the performance curve of the low-level executor. We also observe that including information too far into the future can be detrimental to both training and testing performance. Furthermore, for applications such as real-time video streams, it is not possible to generate future frames for $\tau > 1$. Therefore, in section **??**, we always set $\tau = 1$, which is essentially an inverse dynamics controller. In figure **??** (c), we also note that in contrast with single-motion or few motion trainings such as [**??**], where narrow (layer width 256, 512 for example) neural networks with 2 hidden-layers are used, our algorithm requires much wider and deeper neural network structures. Empirically, we use 3 layers of dimension 1024, trained with 4096 agents, due to limitations of computational resources.

In figure **??**, we show that the motion balancer and the variance controller are essential components to the success of training. Removing any of the two modules from our algorithm will cause a big performance gap.

8.5 Zero-Shot Robustness

Previously in [**??**], results are shown where the agent can resist perturbation from projectiles, or retarget to models with different weight distributions. However, we argue that this leaves a gap between actual applications and what is demonstrated. More specifically, the retargeting is done by retraining a new model, and the

	Speed-1.6	Speed-1.5	Speed-1.4	Speed-1.3	Speed-1.2	Speed-1.1	Speed-0.9	Speed-0.8	Speed-0.7	Speed-0.6	Speed-0.5	Speed-0.4
DeepMimic	21.5%	20.2%	21.5%	22.5%	27.4%	38.1%	36.3%	22.2%	15.0%	13.7%	10.8%	8.1%
Ours	65.8%	76.9%	88.7%	95.9%	98.5%	98.0%	99.2%	96.6%	91.7%	70.0%	47.4%	34.5%
	heavy	light	Proj-1/1Hz	Proj-1/5Hz	Proj-1/10Hz	Proj-1/20Hz	Proj-1/30Hz	Proj-1/40Hz	Proj-1/50Hz	Proj-1/60Hz	Proj-1/70Hz	Proj-1/80Hz
DeepMimic	65.7%	33.5%	13.2%	27.7%	37.5%	51.7%	60.0%	63.0%	62.0%	65.2%	63.9%	68.8%
Ours	77.9%	98.2%	31.6%	69.9%	86.8%	95.9%	97.1%	98.0%	98.7%	98.7%	98.9%	99.2%

Table 2. In this table we show the zero-shot robustness of our algorithm compared with DeepMimic [**?**]. The keyword Speed represents the experiments where we modify the speed of the reference motion with certain ratio, and `Proj` represents the experiments where we modify how frequently the projectiles are thrown at the agents (i. e. how many time-steps there are between two projectiles are thrown at the agent). `heavy` and `light` represent the two experiments where we use different humanoid models. In the table we show the relative performance compared to the original performance.

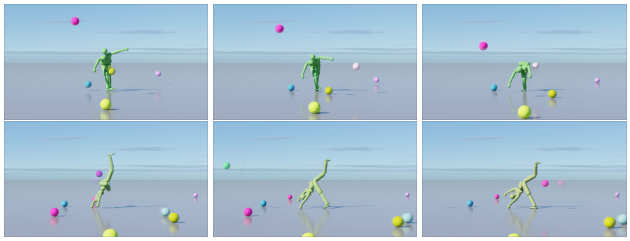

Fig. 12. Snapshots of our agents under projectiles. We refer the readers to the attached videos for more details.

robustness to projectiles, as we show later in the experiments, is still very limited.

In this project, we introduce **zero-shot robustness**, where the agent never sees the perturbation or retargeting information during training, and is asked to perform tasks under perturbations or using different humanoid models with varying masses. We argue that a robust controller with the ability to combat unseen perturbations and retargeting problems will have a much broader potential for real-life applications. More specifically, we have the following zero-shot robustness tasks:

Zero-shot Pertubation Robustness: In this case, the agents are trained without projectiles being thrown at them. During testing, the agents are required to perform the tasks under projectiles. We include experiments with a full range of projectile density and frequencies, and use the reward as the metric to numerically evaluate the performance.

Zero-shot Speed Robustness: Traditionally the motion data samples have a certain fixed speed. In DeepMimic, a phase variable is used to encode the time information of the motion. In this experiment, the agents are trained with motions at the original speed, but during testing, the agents are required to reproduce motions at different speeds.

Zero-shot Model Retargetting Robustness: In this case, instead of retraining on the new humanoid models as in DeepMimic, we ask the agent to reproduce the motion using a different humanoid model, which it has never seen during training. We use humanoid models that are 25% lighter and 25% heavier.

To numerically analyze the performance, we use the relative performance compared to the original performance without perturbation, projectiles or model-mismatch. As we can see from table **??**, our method performs significantly better than DeepMimic. DeepMimic demonstrates very limited zero-shot robustness, while our method can still obtain almost 95% performance in the majority of experiments. We did not include PD-based method and [**?**] in this comparison, as their performance on the evaluated cartwheel motion has a large gap compared to DeepMimic and our method, where they fail to reproduce the specific cartwheel motion when training with a large dataset. We also refer to the attached videos for more details.

9 INTERACTIVE APPLICATIONS

In this section, we combine different high-level motion schedulers with our trained low-level executor (section **??**), showcasing a number of interactive applications. We emphasize that all of the presented applications are real-time interactive and do not require any additional training or fine tuning of our low-level executor, which can be used on-the-fly in all of these settings. Since snapshots cannot fully demonstrate motion, we refer readers to the demo video.

9.1 Keyboard Driven Interactive Control

Here we show the application using the high-level planner described in section **??**. Figure **??** shows the snapshots of our agents controlled by the keyboard command (inset), with our agent shown in yellow and PFNN reference motion shown in white on the left. Note that we do not use inverse kinematics to force the agent's feet to touch the ground in PFNN. This increases the engineering efficiency of PFNN module, and yet we show that the yellow agent generated by UniCon can still demonstrate realistic physics-based motions in a real-time fashion. We are almost able to perfectly follow the target states generated by PFNN.

Due to varying engines and humanoid models, we cannot fully reproduce the original PFNN with more motion gaits and uneven terrain. However, our universal framework's efficiency and simplicity indicates that the motion quality and variety generated by our algorithm is only bottlenecked by the quality of the high-level scheduler used. Our algorithm has the potential to adapt to different schedulers with varying designs and implementations, such as the ones used in basketball and soccer video games.

9.2 Interactive Motion Stitching

In this section, we discuss the results where we randomly select a motion from a motion dataset and our algorithm will respond

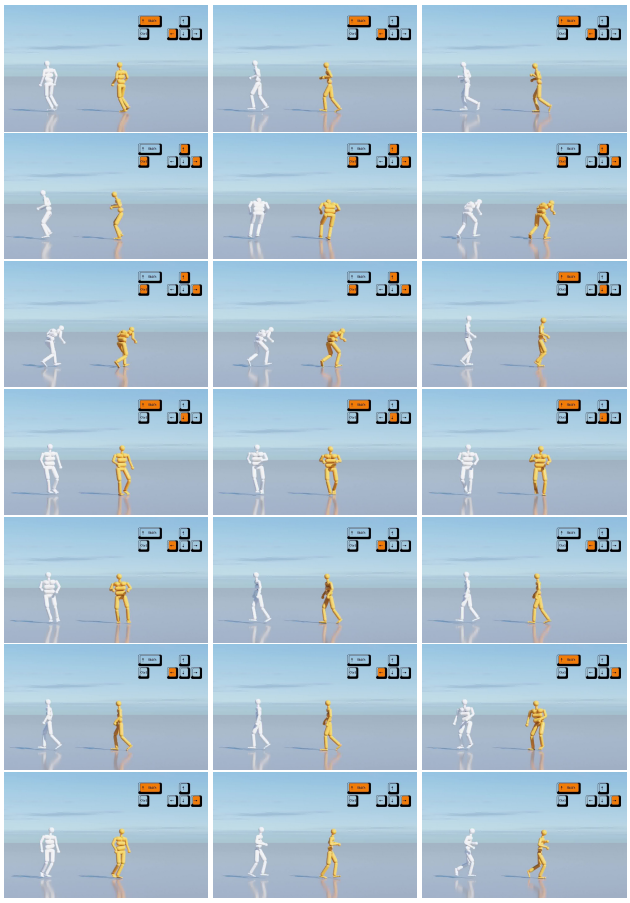

Fig. 13. Snapshots of the keyboard driven interactive control application. Note that our controller is real-time responsive to keyboard commands. On the left is the target agent states, and on the right, the yellow agents represent ones that are physically simulated by our algorithm.

to that real time. In the cover image figure **??**, the agent demonstrates the master of much more interactive complex skills compared to DeepMimic. In the attached video, UniCon also demonstrates strong **emergent physics-based transition**, where we show that, in contrast with some existing methods which design or interpolate realistic transitions, our system can smoothen sharp transition, where animation principles such as anticipation, ease-in & ease-out, are automatically satisfied with our low-level executor. We also note that the transition skills are not recorded in the dataset (not learnt from motion data), and they are mastered by UniCon by generalizing from other skills.

In figure **??**, we further demonstrate the effectiveness and extreme transferability of our algorithm, by forcing our agent to react to motions it has never seen before. Our agent can still generate high-quality physics-based animation as shown in the figure **??**.

It is worth mentioning that motion stitching can be viewed as the simplest motion graph system, which completely ignores generating smooth transitions between motions from the scheduler. However, the low-level executor is able to naturally generate smooth physics-based transitions on its own. Since our algorithm can demonstrate realistic motions using the simplest motion graph system, we believe it can also utilize better designed motion graph systems vastly available both in the research and engineering communities. The transferability skills on unseen motions demonstrated by our model also suggest potential use for motion systems with large motion variety.

9.3 Interactive Video Controlled Animation

In figure **??**, we show how our algorithm can be used to teleport the motions captured from a remote host, to its physics-based avatars in the simulated environment real-time. Note that different from [**?**], our system is real-time and does not require a high-fidelity pose-estimator, or hours of online re-training. In figure **??**, indoor behaviors such as walking, turning, waving and jumping can be animated efficiently. Despite the mismatch in frame rate between the pose-estimator and our simulated environment, and the visually obvious pose estimation errors, our agent generates realistic real-time physics-based motions.

We expect the algorithm can be further improved, and combined with the vast available online datasets generated from video websites such as YouTube.

10 CONCLUSION AND DISCUSSION

In this paper, we proposed a universal neural controller for a variety of real-time interactive control applications. We closely study physics-based motion control on a large-scale dataset where novel techniques including constrained multi-objective reward optimization, motion balancing, and variance control are essential for the success of our framework. The controller we propose obtains much better robustness and generalization compared with existing research, where training and testing are generally performed on the same motion distribution. Once trained, our method does not require further online retraining and can be applied on-the-fly to various applications, such as real-time interactive control from keyboard, videos and motion stitching.

We also identify limitations of our framework and potential research topics for the future. One such topic is exploring how we can further improve the capacity of the neural controller, so that it can master even more skills without worrying about asymptotic performance drop for each motion. Most of the applications we demonstrate are research driven. The high-level schedulers used still have areas for improvement compared with those used in high-quality video games today. It remains to be explored how well our framework can be combined with an AAA video-game motion system. We also did not explore the ability of our method on varying terrain types and physical interactions with objects in the scene. All of these topics present an exciting avenue for future work.

REFERENCES

Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020. Skeleton-Aware Networks for Deep Motion Retargeting. *arXiv preprint arXiv:2005.05732* (2020).

Shailen Agrawal and Michiel van de Panne. 2016. Task-based locomotion. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 82.

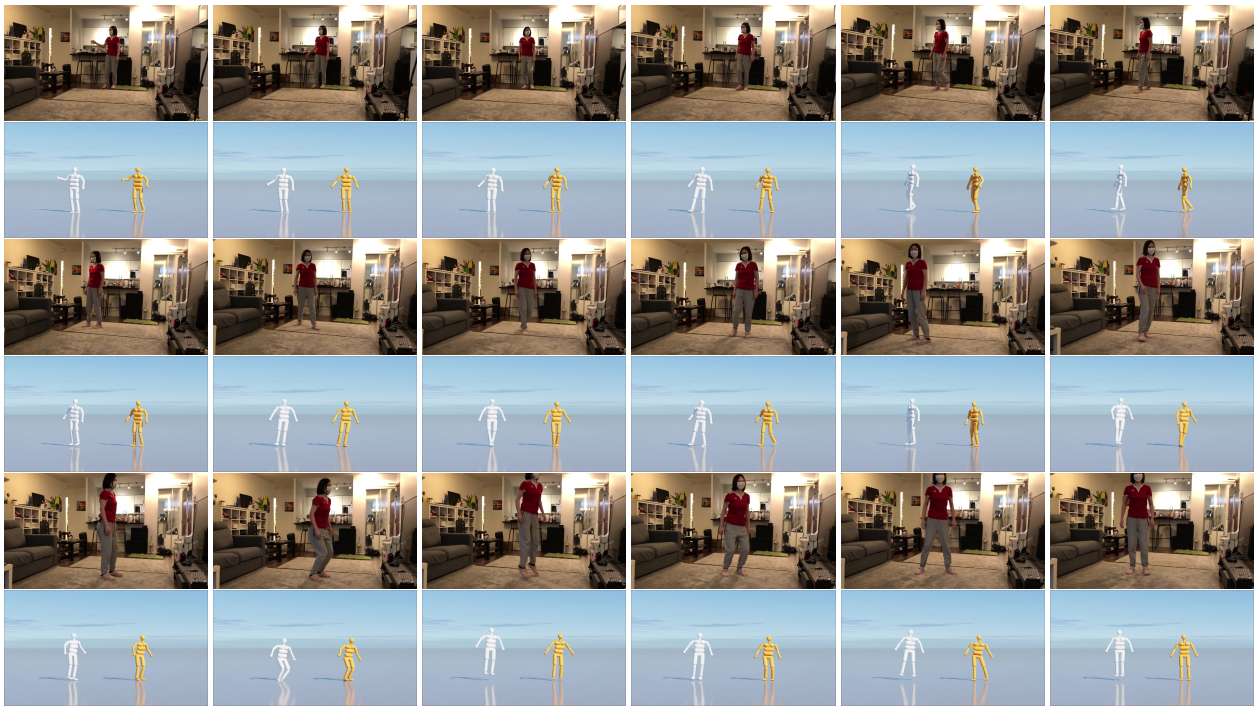

Fig. 14. In this figure, we show how our controller reacts real-time to the remote host captured by a camera. It successfully reproduces waving, walking, turning and jumping behaviors.

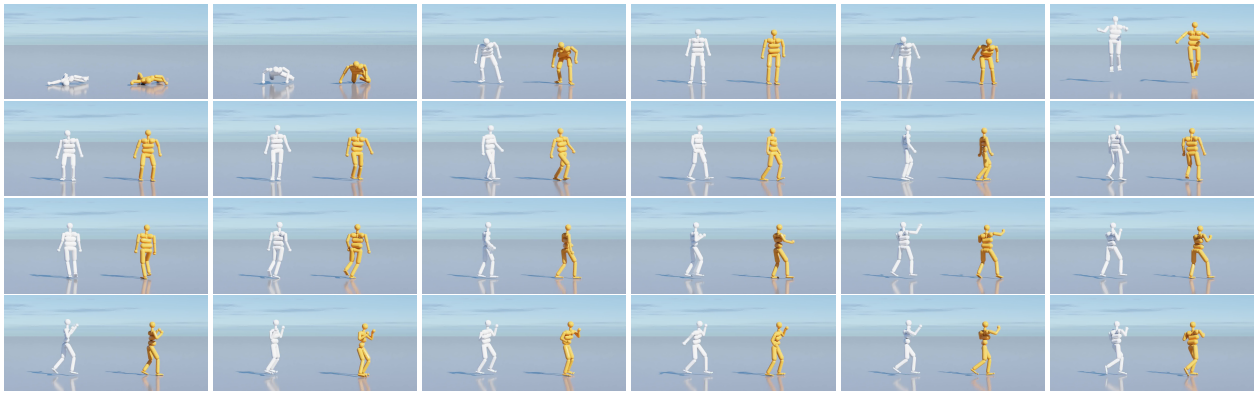

Fig. 15. Motion stitching scheduler performance on unseen motions. Our controller can react real-time to generate getting up, walking and boxing behaviors.

Okan Arikan and David A Forsyth. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 483–490.

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: data-driven responsive control of physics-based characters. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–11.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540* (2016).

Michael Buttner. 2015. Motion Matching-The Road to Next-Gen Animation. In *Nucl. ai Conference*.

Michael Buttner. 2019. Machine Learning for Motion Synthesis and Character Control. In *Interactive 3D Graphics and Games (I3D) 2019*. https://www.youtube.com/watch?v=zuvmQxcCOM4

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.

Simon Clavet. 2016. Motion matching and the road to next-gen animation. *Proc. of GDC 2016* (2016).

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

Haegwang Eom, Daseong Han, Joseph S Shin, and Junyong Noh. 2019. Model Predictive Control with a Visuomotor System for Physics-based Character Animation. *ACM Transactions on Graphics (TOG)* 39, 1 (2019), 1–11.

Christoph Feichtenhofer. 2020. X3D: Expanding Architectures for Efficient Video Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 203–213.

Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive character animation using simulated physics: A state-of-the-art review. In *Computer graphics forum*, Vol. 31. Wiley Online Library, 2492–2515.

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.

Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. 2017. Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138* (2017).

Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers*. 522–531.

Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. 2018. DensePose: Dense Human Pose Estimation In The Wild. *arXiv preprint arXiv:1802.00434* (2018).

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).

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.

Geoff Harrower. 2018. Real Player Motion Tech in EA Sports UFC 3. In *Proc. of GDC 2018*.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. 2016. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182* (2016).

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

Jessica Hodgins. 2015. CMU graphics lab motion capture database.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 42.

Umar Iqbal, Pavlo Molchanov, and Jan Kautz. 2020. Weakly-Supervised 3D Human Pose Learning via Multi-view Images in the Wild. *arXiv preprint arXiv:2003.07581* (2020).

Angjoo Kanazawa, Michael J Black, David W Jacobs, and Jitendra Malik. 2018. End-to-end recovery of human shape and pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7122–7131.

L Kovar. 2002. Motion graphs. *ACM Trans. Graph.* 21, 3 (2002), 473–482.

Lucas Kovar and Michael Gleicher. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics (ToG)* 23, 3 (2004), 559–568.

Taesoo Kwon, Yoonsang Lee, and Michiel Van De Panne. 2020. Fast and flexible multilegged locomotion using learned centroidal dynamics. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 46–1.

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 491–500.

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–10.

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion Fields for Interactive Character Locomotion. In *ACM SIGGRAPH Asia 2010 Papers* (Seoul, South Korea) *(SIGGRAPH ASIA '10)*. Association for Computing Machinery, New York, NY, USA, Article 138, 8 pages. https://doi.org/10.1145/1866158.1866160

Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research* 18, 1 (2017), 559–563.

Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–10.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. 2018. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. *arXiv preprint* (2018). arXiv:1810.05762 https://arxiv.org/abs/1810.05762

Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel Van De Panne. 2020. Character controllers using motion VAEs. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 40–1.

Chunming Liu, Xin Xu, and Dewen Hu. 2014. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 3 (2014), 385–398.

Libin Liu and Jessica Hodgins. 2017. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 1–14.

Libin Liu and Jessica Hodgins. 2018. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Libin Liu, Michiel Van De Panne, and KangKang Yin. 2016. Guided learning of control graphs for physics-based characters. *ACM Transactions on Graphics (TOG)* 35, 3 (2016), 1–14.

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving Sampling-based Motion Control. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 415–423.

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based contact-rich motion control. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 128.

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

Miles Macklin, Kenny Erleben, Matthias MÃijller, Nuttapong Chentanez, Stefan Jeschke, and Viktor Makoviychuk. 2019. Non-smooth Newton Methods for Deformable Multi-body Dynamics. *ACM Transactions on Graphics* 38 (10 2019), 1–20. https://doi.org/10.1145/3338695

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. 2018a. Hierarchical visuomotor control of humanoids. *arXiv preprint arXiv:1811.09656* (2018).

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. 2018b. 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).

Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. 2020. Catch & Carry: reusable neural controllers for vision-guided whole-body tasks. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 39–1.

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++ a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–12.

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–11.

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 Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. 2019. MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. *arXiv preprint arXiv:1905.09808* (2019).

Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. 2018b. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–14.

Xue Bin Peng and Michiel van de Panne. 2017. Learning locomotion skills using deeprl: Does the choice of action space matter?. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–13.

Anil V Rao. 2009. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences* 135, 1 (2009), 497–528.

Arthur George Richards. 2005. *Robust constrained model predictive control*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH 2007 papers*. 106–es.

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

Dian Shao, Yue Zhao, Bo Dai, and Dahua Lin. 2020. Finegym: A hierarchical video dataset for fine-grained action understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2616–2625.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural state machine for character-scene interactions. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

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

Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning* 84, 1-2 (2011), 51–80.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. 2019. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057* (2019).

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 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach,

R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5320–5329. http://papers.nips.cc/paper/7116-robust-imitation-of-diverse-behaviors.pdf

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A scalable approach to control diverse behaviors for physically simulated characters. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 33–1.

Jia-chi Wu and Zoran Popović. 2010. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 1–10.

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.

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