Physics-based Character Controllers Using Conditional VAEs

JUNGDAM WON, Meta AI, USA
DEEPAK GOPINATH, Meta AI, USA
JESSICA HODGINS, Meta AI / Carnegie Mellon University, USA

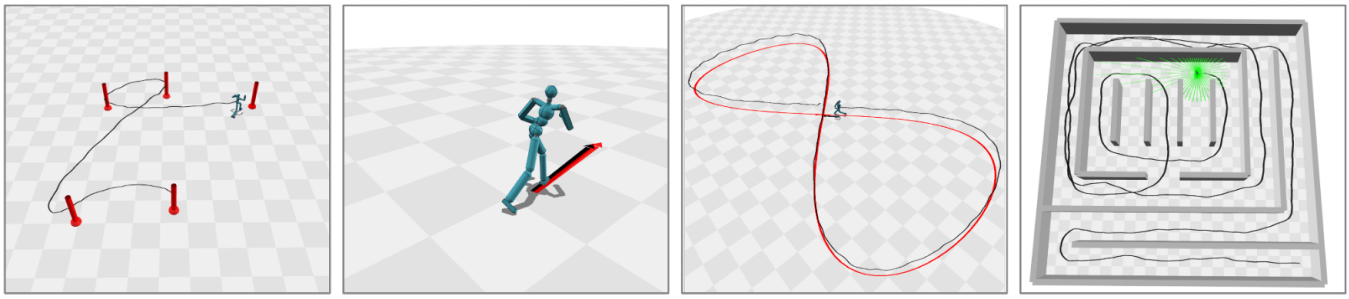


Fig. 1. Given an input motion capture dataset, conditional VAEs are used to learn a controller for physically simulated characters. The controller can perform behaviors that are similar to but not identical to the input without conditioning on a specific goal. A variety of downstream tasks can then be solved efficiently based on the pre-trained controller such as *Point-Goal Navigation*, *Joystick Control*, *Path Follower*, and *Maze Runner*.

High-quality motion capture datasets are now publicly available, and researchers have used them to create kinematics-based controllers that can generate plausible and diverse human motions without conditioning on specific goals (i.e., a task-agnostic generative model). In this paper, we present an algorithm to build such controllers for physically simulated characters having many degrees of freedom. Our physics-based controllers are learned by using conditional VAEs, which can perform a variety of behaviors that are similar to motions in the training dataset. The controllers are robust enough to generate more than a few minutes of motion without conditioning on specific goals and to allow many complex downstream tasks to be solved efficiently. To show the effectiveness of our method, we demonstrate controllers learned from several different motion capture databases and use them to solve a number of downstream tasks that are challenging to learn controllers that generate natural-looking motions from scratch. We also perform ablation studies to demonstrate the importance of the elements of the algorithm. Code and data for this paper are available at: https://github.com/facebookresearch/PhysicsVAE

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Motion capture**; *Reinforcement learning*; *Learning from demonstrations*.

Additional Key Words and Phrases: Character Animation, Physics-based Simulation and Control, Motion Capture, Reinforcement Learning, Variational Autoencoder, Behavior Cloning

Authors' addresses: Jungdam Won, Meta AI, Pittsburgh, USA, jungdam@fb.com; Deepak Gopinath, Meta AI, Pittsburgh, USA, dgopinath@fb.com; Jessica Hodgins, Meta AI / Carnegie Mellon University, Pittsburgh, USA, jkh@fb.com.

0730-0301/2022/7-ART96
https://doi.org/10.1145/3528223.3530067

ACM Reference Format:
Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2022. Physics-based Character Controllers Using Conditional VAEs. *ACM Trans. Graph.* 41, 4, Article 96 (July 2022), 12 pages. https://doi.org/10.1145/3528223.3530067

1 INTRODUCTION

Recently, publicly available motion capture datasets [CMU 2002; Harvey et al. 2020; Mahmood et al. 2019; Tsuchida et al. 2019] have been used to learn machine learning models that can produce plausible human motions. Kinematics-based controllers based on generative models such as variational autoencoders (VAEs) [Ling et al. 2020] or normalizing flow [Henter et al. 2020] have been developed to model many variations of common behaviors existing in these large datasets. Once learned, those models can generate full-body human motions without conditioning on specific goals and the generated motions resemble the original motions in the input database. More specifically, they output a pose (i.e., a set of skeletal joint angles) at the next timestep given both the current pose and a latent random vector sampled from a fixed random distribution (e.g., the standard normal distribution), this process can repeat to generate a long sequence. Once learned, the controller can play the role of a manifold for generating plausible human motions when learning other downstream tasks such as motion prediction or direct control through user-provided inputs.

Physics-based controllers for generating human motions often use large motion capture datasets [Chentanez et al. 2018; Fussell et al. 2021; Merel et al. 2019b; Park et al. 2019; Won et al. 2020]. Some controllers generate physically simulated motions that are almost indistinguishable from the ground truth motion capture data. However, most approaches either learn pure imitation controllers or are based on non-generative models, so the learned controller

requires the original reference motion capture data as input to generate plausible motions at runtime. Furthermore, the transfer to new downstream tasks requires extensive reward engineering to maintain motion quality.

The goal of this paper is to build controllers for physically simulated characters, that have similar functionality to what has been shown in kinematics-based controllers [Henter et al. 2020; Ling et al. 2020]: (a) generate motions that look natural and resemble the training data without conditioning on specific goals, (b) reuse in many downstream tasks without complex reward engineering. If we have physics-based controllers equipped with such functionality, they can be used for a much wider variety of applications than kinematic characters because using physics simulation allows for new scenarios that are not in the training data. For example, new physical interactions, unseen environments, and characters that do not exactly match the motion capture data are all possible with physically simulated characters. However, achieving such functionality is challenging because physical constraints make it more difficult to generate plausible motions. For example, a small error can push the characters to an unrecoverable state such as falling down. To tackle these challenges, we learn a physics-based controller in a supervised manner by using conditional VAEs, where behavior cloning and a differentiable physics simulation layer are combined. Our controller is learned and run without any information on downstream tasks. Once learned, the physically simulated character can transit among different behaviors autonomously by sequentially feeding random latent vectors into the learned controller. The transfer of the pre-trained task-agnostic controller to a specific downstream task can be performed by deep reinforcement learning (deep RL) with a control policy that uses the controller as the low-level motor primitives. We observed that the simulated character equipped with our pre-trained controller could often perform plausible behaviors even in the first learning iteration of deep RL. This capability reduces the exploration burden for deep RL algorithms, allowing the efficient learning of new tasks while generating natural-looking motions. To show the effectiveness of our method, we test our methods with various motion capture databases and several high-level downstream tasks that are challenging to solve from scratch. The contributions of this paper are as follows:

- **Novel Results.** We present a physics-based controller that can generate long sequences of natural-looking motion for high degree-of-freedom bipedal characters without any task-specific inputs. More specifically, the motion can simply be generated by sequentially conditioning random latent vectors sampled from the standard normal distribution. The approach works for a wide variety of behaviors, given appropriate motion capture data for training. This problem is very challenging for bipedal characters because we should consider not only balancing but also naturalness. We test it for several motion capture databases including locomotion, sports, and dance.
- **Novel Technical Components.** We develop a stochastic and generative structure for physically simulated characters based on conditional VAEs, which is enabled by combining behavior cloning and a differentiable physics simulation layer. We further develop an auxiliary method (which we call a helper branch) that aids in effective transfer learning that avoids motion degradation. By using the helper branch, the pre-trained controller produces natural-looking motions for tasks that the input motion capture database does not fully cover.
- **Support for Other Downstream Applications.** We demonstrate the usefulness of our pre-trained controllers by showing that various high-level downstream tasks can be solved efficiently. Because they are fully task-agnostic, applications are not limited to what we demonstrated in this paper. We believe that our pre-trained controllers could provide a solid foundation for developing a variety of physics-based character controllers that are challenging to learn from scratch such as combining monocular-camera motion capture with physics simulations or learning competitive policies for large multi-agent environments.

2 RELATED WORK

We review the prior work that is most closely related to our work in physics-based character control with deep RL. We also review several studies in kinematics-based motion controllers that use deep neural networks because our work is inspired by some approaches using auto-regressive generative models.

2.1 Physics-based Character Controller

Using physics simulation to generate plausible motions for articulated characters has been studied since the 1990's [Hodgins et al. 1995; Laszlo et al. 1996]. This area of research includes motor-actuated humans [Coros et al. 2010; Lee et al. 2010; Liu et al. 2016; Ye and Liu 2010; Yin et al. 2007], muscle-actuated humans [Geijtenbeek et al. 2013; Lee et al. 2019, 2014; Wang et al. 2012], other animals and creatures [Coros et al. 2011; Luo et al. 2020; Peng et al. 2015; Tan et al. 2011a; Won et al. 2017; Wu and Popović 2003]. Recently, the control capability of physically simulated characters has significantly developed because of breakthroughs in deep RL. Liu et al. [2017] proposed controllers trained by deep Q-learning where a short-horizon linear feedback controller was used as a single action unit. Peng et al. [2018] introduced an imitation controller for a single motion clip, where the basic idea is to compute rewards directly by comparing the simulated humanoids with the input motion capture data. It was then extended to multiple motion clips [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2021], large motion capture datasets [Chentanez et al. 2018; Fussell et al. 2021; Merel et al. 2019b; Won et al. 2020]. One of the primary benefits of using deep RL is that its capability can be extended to learn new skills, for example, other types of terrains [Peng et al. 2017; Xie et al. 2020], handling small objects such as balls or boxes [Liu and Hodgins 2018; Merel et al. 2020; Peng et al. 2019], different morphologies [Ryu et al. 2020; Won and Lee 2019], diverse skills [Lee et al. 2021; Yin et al. 2021], or multi-agent scenarios [Haworth et al. 2020; Liu et al. 2021; Won et al. 2021]. Theses methods generated an impressive array of behaviors and demonstrated the value of physical simulation for characters interacting in complex ways with the environment.

2.2 Kinematics-based Character Controller

Kinematics-based controllers can generate plausible human motions automatically based on deep neural networks and large motion capture databases such as [CMU 2002; Harvey et al. 2020; Mahmood et al. 2019]. One popular network model is a fixed-length model, where the entire motion (a sequence of poses) is generated at once. Holden et al. [2016] proposed a feed-forward network model with convolution along the time axis, and the learned manifold can be used for the motion editing and synthesis. Harvey et al. [2020] solved a fixed-duration motion in-betweening problem by using an encoder-decoder LSTM architecture with additional positional encoding and time-to-arrival inputs. Another popular model is an auto-regressive model, where the motion is generated frame by frame. The key challenge in using auto-regressive models is to prevent the model from collapsing or diverging during rollouts. Auxiliary inputs such as phase variables or user-provided control inputs are frequently used to further constrain possible poses at the next timestep [Holden et al. 2017; Lee et al. 2018; Starke et al. 2019, 2020]. For pure auto-regressive models which do not take any extra inputs, incorporating stochastic components into the model resolved the challenge of the uncertainty (or ambiguity) of the future motion. Henter et al. [2020] used normalizing flows while Ling et al. [2020] and Ghorbani et al. [2020] used conditional VAEs, where the uncertainty on next possible poses is encoded by a random distribution conditioned on the latent space and the generated motions can be further conditioned by an input action label. Unlike the models using auxiliary inputs, these models can generate motions in a fully autonomous (i.e., task-agnostic) manner from a random vector sampled from the random distribution used in training the models. This performance demonstrated impressive generalization of the training data. The learned model can also be reused as a base controller when learning other downstream tasks such as motion prediction or direct control through user-provided inputs. For example, a navigation controller that modulates the latent vector of the pre-trained base controller can be learned by using deep RL [Peng et al. 2019].

2.3 Manifold (Latent Space) Learning for Physically Simulated Humanoids

Just as task-agnostic kinematics-based controllers can be used as manifolds to constrain the generated motions to be natural, there have been approaches that attempt to construct manifolds for physically simulated characters. Peng et al. [2019] proposed a multiplicative composite policy composed of the expert policies that are first learned in a task-agnostic manner by imitation rewards. Merel et al. [2019b] performed distillation with an encoder-decoder policy with the *AR(1)* prior by using expert trajectories generated by the pre-trained imitation policies. The model was further fine-tuned for object catching and carrying skills [Merel et al. 2020] and soccer play [Liu et al. 2021]. Won et al. [2021] used a policy composed of a task-encoder and a mixture-of-experts motor-decoder, where the decoder was first learned by an imitation policy similarly to [Peng et al. 2019] then fine-tuned for other multi-agent tasks. Although the pre-trained latent space models mentioned above were able to encapsulate the input motion capture data efficiently, they cannot generate long sequences of natural-looking motions without conditioning on the original reference motions because they are based on a deterministic latent space model. Hasenclever et al. [2020] proposed joint training of imitation and downstream tasks using the shared latent space, where the stochastic latent space model was incorporated. However, the motion quality of the downstream task policies was significantly degraded when compared to the input motions, and the ability to generate long sequences of natural-looking motions was not demonstrated. Our method aims to build models that can autonomously generate long sequences that are similar to the input motion capture data and can be transferred to other downstream tasks with less motion degradation. To the best of our knowledge, such robust models have not been demonstrated for high degrees-of-freedom physically simulated humanoids with a floating base.

3 CONTROLLERS USING CONDITIONAL VAES

Our method for learning physics-based controllers using conditional VAEs can be understood as an extension of behavior cloning that learns an expert's policy (e.g., a mapping from state to action) given input expert trajectories $\{\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \cdots, \mathbf{s}_T\}_1^N$, where $\mathbf{s}_t$ and $\mathbf{a}_t$ are the state and action at time t, respectively, and N is the total number of trajectories. In our application, the expert trajectories can be given by any existing controllers such as deep imitation controllers [Bergamin et al. 2019; Fussell et al. 2021; Park et al. 2019; Peng et al. 2018, 2021; Won et al. 2020] or traditional linear-feedback controllers [Coros et al. 2009; Lee et al. 2010; Ye and Liu 2010; Yin et al. 2007]. The basic loss function for behavior cloning is the sum of squared error $\sum_K \|\mathbf{a}_i - c(\mathbf{s}_i)\|^2$, where K is the total number of state-action pairs included in the entire trajectory and $c(\cdot)$ is a learnable control policy, which is a deep neural network in our case. Once the policy is learned in a supervised manner, the simulation updates the state sequentially (i.e., rollouts) while the actions are computed from the policy during runtime.

One of the primary challenges in behavior cloning is that the state can easily drift out-of-distribution because approximation error is accumulated during rollouts. Physically simulated humanoids are brittle to such errors, for example, a small prediction error on the foot landing position can easily make the simulated humanoid lose balance. Naïve behavior cloning using the basic loss function does not perform well for humanoid characters in general because the function only considers errors occurred in a single timestep. To tackle this challenge, we develop a conditional VAE [Sohn et al. 2015] to further constrain (or regularize) the predicted action. Figure 2 illustrates our conditional VAE structure, where it takes the current and next states $(\mathbf{s}_t, \mathbf{s}_{t+1})$ as inputs then reconstructs the next state $\mathbf{s}'_{t+1}$, where both the encoder and the decoder are conditioned by the current state $\mathbf{s}_t$. The conditional VAE decoder includes two subcomponents, the differentiable physics simulation layer (i.e., the function from the current state, action to the next state) and the motor-decoder. During the decoding process, the action $\mathbf{a}'_t$ is first computed by the motor-decoder then it is passed to the simulation layer to predict the next state $\mathbf{s}'_{t+1}$ induced by the given action. The simulation layer is pre-trained by supervision with the input expert trajectories and the weights are frozen when learning the

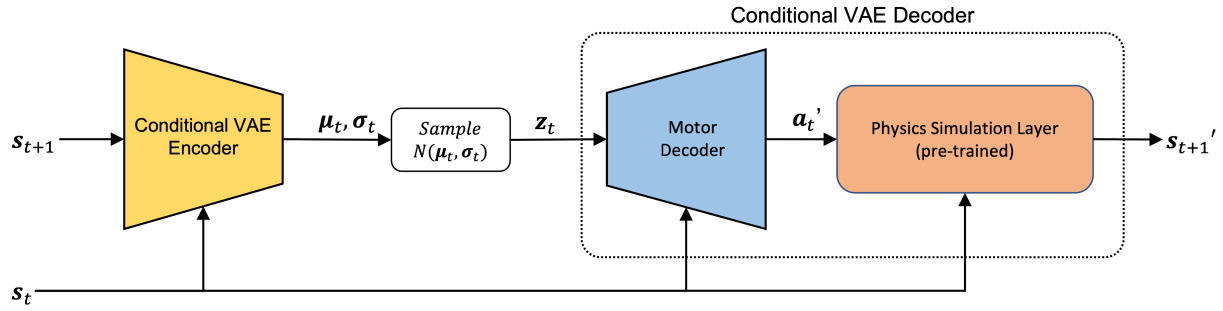


Fig. 2. A conditional VAE structure for learning controllers of physically simulated characters

other components in the conditional VAE. Once it is learned, we can use the motor-decoder to generate motions by simply conditioning both the current state $\mathbf{s}_t$ and a random latent vector $\mathbf{z}_t$ at each timestep, where $\mathbf{z}_t$ is sampled from the standard normal distribution $N(\mathbf{0}, \mathbf{I})$ of which mean and covariance matrix are zero and identity, respectively.

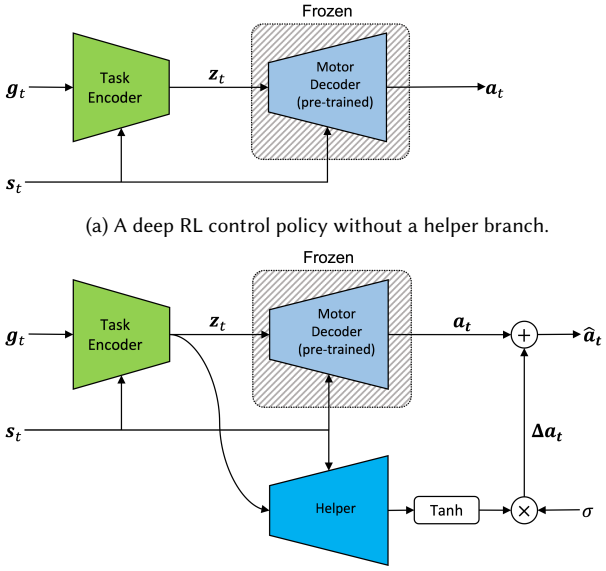
We use the state representation $\mathbf{s}_t = (\mathbf{p}_t, \mathbf{q}_t, \mathbf{v}_t, \mathbf{w}_t)$, where $\mathbf{p}_t \in \mathbb{R}^{Jx3}$, $\mathbf{q}_t \in \mathbb{R}^{Jx6}$, $\mathbf{v}_t \in \mathbb{R}^{Jx3}$, and $\mathbf{w}_t \in \mathbb{R}^{Jx3}$ are positions, orientations, linear velocities, and angular velocities of all links of the simulated humanoid at time t, respectively, and are represented with respect to the current facing transformation, which is defined by the facing direction, the gravity direction, and the root position projected on the ground [Won et al. 2020]. Note that we use a 6-dof representation for the orientation, which uses the first two columns of the rotation matrix. Our simulated humanoid is equipped with stable PD (proportional derivative) controllers [Tan et al. 2011b] and the action space $\mathbf{a}_t$ is a set of target joint angles for the controllers where the value in each dimension is bounded by $[-3\ rad, 3\ rad]$. The loss function we use to train the conditional VAE is

$$\sum_N \left[\|\mathbf{a}_t - \mathbf{a}'_t\|^2 + \alpha \cdot \|\mathbf{s}_{t+1} - \mathbf{s}'_{t+1}\|^2 + \beta \cdot D_{KL}\big(N(\mu_t, \sigma_t) \parallel N(\mathbf{0}, \mathbf{I})\big) \right] \quad (1)$$

where $(\mathbf{s}_t, \mathbf{a}_t)$ is a state-action pair in the input expert trajectories, $(\mathbf{s}'_t, \mathbf{a}'_t)$ is a state-action pair generated from the conditional VAE, $D_{KL}(\cdot \parallel \cdot)$ measures the KL-divergence of the two distributions, μ_t, σ_t are the mean and standard deviation generated from the encoder, and α, β are relative weights of the terms. The first term is a typical behavior cloning loss, the second self-supervision term further regularizes the output action $\mathbf{a}'_t$ so that the motor-decoder considers the future results by the gradient signals propagated through the simulation layer (i.e., preventing the next state $\mathbf{s}'_t$ induced by the current output action $\mathbf{a}'_t$ from being out-of-distribution), and the third term allows us to use the motor-decoder as a generative model which we can generate motions via simulation rollouts in a task-agnostic manner.

4 LEARNING DOWNSTREAM TASKS

We use deep RL to learn control policies for various downstream tasks. At each time step t, the agent (the simulated character in our case) observes its environment through the proprioception state $\mathbf{s}_t$ and the goal state $\mathbf{g}_t$, and performs an action $\mathbf{a}_t$. Then, the states



(a) A deep RL control policy without a helper branch.

(b) A deep RL control policy with a helper branch.

Fig. 3. Control policy structures using a pre-trained motor-decoder. During control policy learning, the weights of the pre-trained motor-decoder are frozen while the remaining weights are updated.

change to the new states $\mathbf{s}_{t+1}$, $\mathbf{g}_{t+1}$ and the agent receives a scalar-valued signal $r_t = r(\mathbf{s}_t, \mathbf{g}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}_{t+1})$, where high rewards mean that the performed action and the state change were desirable. In deep RL, we use a control policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g}_t)$ represented by a deep neural network to compute the actions, where $\pi_\theta(\cdot|\cdot)$ is the probability of taking actions under the given states and θ is the network weights. The goal of deep RL is to find an optimal policy that maximizes the expected returns $J(\theta) = E\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]$, where $\gamma \in (0, 1)$ is a discount factor that considers how far into the future will be taken into account. The formulation can also be further extended for multiple agents based on a partially observable Markov game, please refer to [Littman 1994] for the details.

Figure 3a illustrates how we construct a deep RL control policy by using a pre-trained motor-decoder, where a task-encoder modulates the latent vector $\mathbf{z}_t$ according to the task-specific observation (i.e.,

the goal state $\mathbf{g}_t$). During control policy learning, we update the task-encoder while freezing the weights of the motor-decoder. We observed that the deep RL control policy equipped with the pre-trained motor-decoder could generate plausible motions even in the first learning iteration. This property enables deep RL algorithms to learn complex and high-level behaviors efficiently while generating natural-looking motions.

Using a control policy like Figure 3a guarantees decent motion quality with reasonable task performance when the input expert trajectories include complete motor skills for the target downstream task. However, the task performance could be degraded if the trajectories are not a good match, for example, if we are attempting to learn a navigation skill for uneven terrain using only walking motions recorded on flat ground. A quick remedy is to update the pre-trained motor-decoder (i.e., end-to-end learning). This approach would provide better task performance because the motor skills that the motor-decoder generates can be adapted for the situations that are not well covered by the input trajectories. However, this adaptation could cause a *forgetting* problem where the motions generated by the motor-decoder are no longer similar to the motions in the input expert trajectories and appear unnatural when compared to the original motions. There always exists a trade-off between *adaptability* and *naturalness* when reusing a pre-trained motor-decoder for a new task, the *learnable motor-decoder* and the *frozen motor-decoder* methods are two extremes.

We develop a hybrid method that allows the policy to adapt to new environments while maintaining the original motion style (or naturalness) and allows users to explicitly tune the degree of adaptation. Figure 3b shows a control policy structure, where we use an additional branch that gets both the current proprioception state $\mathbf{s}_t$ and the latent vector $\mathbf{z}_t$ as inputs then outputs the action offset $\Delta\mathbf{a}_t$, which is then added to the pre-trained motor-decoder output $\mathbf{a}_t$. The sum $\hat{\mathbf{a}}_t = \mathbf{a}_t + \Delta\mathbf{a}_t$ becomes an action for our simulated humanoid, more specifically, it is computed as follows:

$$\Delta\mathbf{a}_t = \sigma \otimes Tanh(H(\mathbf{s}_t, \mathbf{z}_t)) \tag{2}$$

where $Tanh$ is the element-wise hyperbolic tangent function, $\otimes$ is the element-wise multiplication, H is a function composed of learnable layers in the additional branch which we call *helper*, and $\sigma = (\sigma_1, \sigma_2, \cdots)$ is a set of user-specified parameters that controls the range of the action offset where the i-th range is confined by $(-\sigma_i, \sigma_i)$ accordingly. The original target joint angles $\mathbf{a}_t$ generated by the motor-decoder are modulated only as much as allowed by the parameters σ, which can be manually specified by the user. The policy will be updated in a similar fashion to the *end-to-end* method if the parameters are very large because the helper can dominate the output of the motor-decoder, whereas it will be updated similarly to the *frozen motor-decoder* when the parameters are near zero because the helper does not significantly affect the action output. The primary benefit of this structure is that users can explicitly tune the allowable motion adaptation (or degradation) depending on the input dataset, the target downstream task, and their preference. For example, when the input expert trajectories are incomplete, users can achieve both plausible motions and acceptable task performance by gradually increasing the degree of adaption.

Table 1. Parameters used in our experiments

Physics Simulation	Simulation rate (Hz)	480
	Control rate (Hz)	30
Deep RL (DD-PPO)	Learning rate (α_π)	$2.0e^{-5}$
	Discount factor (γ)	0.99
	GAE and TD (λ)	0.95
	# tuples per update (T)	50000
	Iteration per update (N)	20
	Batch size (n)	256
	Clip parameter (ϵ)	0.2
Conditional VAE Parameters	α	$1.0e^{-4}$
	β	2.0
	Latent dimension	32

Table 2. Deep Neural Network Structure

Conditional VAE Encoder	Type	MLP
	Depth (layers)	2
	Width (hidden units)	(256, 128)
	Activation	(ReLu, ReLu, Linear)
Motor Decoder	Type	MLP
	Depth (layers)	3
	Width (hidden units)	(512, 512, 512)
	Activation	(ReLu, ReLu, ReLu, Linear)
Physics Simulation Layer	Type	MLP
	Depth (layers)	2
	Width (hidden units)	(1024, 1024)
	Activation	(ReLu, ReLu, Linear)
Task Encoder	Type	MLP
	Depth (layers)	2
	Width (hidden units)	(256, 128)
	Activation	(ReLu, ReLu, Linear)
Helper	Type	MLP
	Depth (layers)	2
	Width (hidden units)	(128, 128)
	Activation	(ReLu, ReLu, Linear)

5 RESULTS

Our environments were implemented based on a publicly available framework [Won et al. 2020], which uses PyBullet [2019], PyTorch [2019], and RLlib [2018] for physics simulation, deep neural network evaluation, and deep RL algorithms, respectively. The movable joints in the simulated humanoid are modeled by ball joints, which are controlled by stable PD servos given a target posture as an input. Table 1 summarizes all parameters related to physics simulation, deep RL, conditional VAEs and Table 2 includes information on the deep neural network structures that we use in the experiments. We use a workstation equipped with 2 CPUs (Intel Xeon CPU E5-2698 v4) and 2 GPUs (Quadro GP100) when learning controllers using conditional VAEs, and combine the same four machines together when learning deep RL control policies for the downstream tasks.

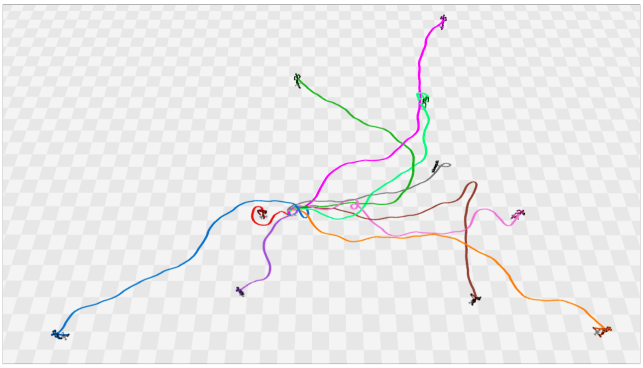


Fig. 4. Ten rollouts using the controller learned with the locomotion dataset. All characters were initialized with the same initial state and simulated for 30 seconds. The size of the grid is 1m by 1m.

5.1 Controllers using conditional VAEs

5.1.1 Expert Trajectory Generation. Because there is no existing dataset that provides expert trajectories (state-action pairs) for simulated humanoid characters, we create our own datasets using existing motion capture data in every experiments. More specifically, given motion capture data, we learn imitation controllers for the motion clips by using deep RL. We use DeepMimic-style [Peng et al. 2018] imitation controllers with multiplicative rewards [Won et al. 2020], and the controllers are learned by DD-PPO [Wijmans et al. 2020]. State-action pairs are collected through simulation rollouts, we repeat the rollouts ten times for each motion clip so that the generated data cover the state and the action spaces widely. Similar to previous studies, our imitation controllers also use a multivariate Gaussian policy with a constant diagonal covariance matrix $\epsilon\mathbf{I}$, where we use $\epsilon = 0.05$ when collecting state-action pairs.

5.1.2 Learning. Given expert trajectories, constructing a controller requires two supervised learning stages: learning a differentiable physics simulation layer and the entire conditional VAE structure. For both cases, we apply a straightforward supervised learning procedure with the Adam optimizer for weight updates, 256 batch size for SGD computation, decaying the initial learning rate 0.01 by 0.7 every 50 epochs. The maximum training epochs that we use are 300 and 600, respectively, the training time usually takes 3-4 hours for the simulation layer, one day for the entire conditional VAE structure. Note that we only use CPUs for the computation.

5.1.3 Random Rollouts. We demonstrate a controller learned with a locomotion dataset generated by PFNN [2017] because locomotion is essential for many high-level downstream tasks such as navigation or joystick control. The total length of the motions that we use is 10 minutes, which includes walking, running, and crouching with various speeds and directions on the flat ground. By following the procedure mentioned in section 5.1.1, the raw motion clips are converted into expert trajectories that include 180,000 state-action pairs (600 seconds x 30 FPS x 10 repetitions).

Figure 4 shows the motions generated by ten rollouts, where all the characters started from the same initial state (a standing posture with zero velocity located at the center of the scene). More specifically, at every timestep, we randomly sample a latent vector $\mathbf{z}_t$ from the standard normal distribution whose mean and covariance matrix are zero and identity, respectively, compute an action $\mathbf{a}_t$ through the motor-decoder, and update the state of the character through the physics simulator. This rollout lasts for 30 seconds and we repeat this ten times. Although all the generated motions resemble the input locomotion dataset, their actual trajectories are all different, which shows that the controller successfully learned not only the style but also the variance existing in the dataset. Furthermore, the fact that the motions generated by our controller cover a wider repertoire than the input expert trajectories implies that generalization was also achieved to some extent. In other words, our controller can transit between different behaviors such as walking, running, and crouching in an arbitrary order without conditioning any extra input except for $\mathbf{s}_t$ and $\mathbf{z}_t$.

5.2 Baseline Downstream Tasks

To understand how effective the pre-trained controllers are in learning various downstream tasks, we first test the four baseline downstream tasks shown in Motion-VAEs [2020]. Please note that their results are kinematic whereas we generate similarly plausible behaviors for physically simulated characters. The four tasks are *Point-Goal Navigation*, *Joystick Control*, *Path Follower*, and *Maze Runner*, we use the helper branch with $\sigma = (0.1, 0.1, \cdots)$ for every task, and the details for each environment are described below:

Point-Goal Navigation. The task is to reach a goal point on the ground and we regard the task as being completed when the distance between the root joint position projected on the ground and the goal point is less than 0.5 meter (see Figure 1a). The reward function we use is $r_t = (d_{t-1} - d_t)$, where d_{t-1} and d_t are distances to the goal point in the previous and the current timesteps. The goal state $\mathbf{g}_t = (\frac{\mathbf{x}_{rel}}{\|\mathbf{x}_{rel}\|}, \|\mathbf{x}_{rel}\|)$ includes the direction and the distance to the goal point, where $\mathbf{x}_{rel} = \mathbf{R}_{face}^{-1}(\mathbf{p}_{goal} - \mathbf{p}_{face})$ is the relative location of the goal point $\mathbf{p}_{goal}$ with respect to the current facing transformation of which rotation and position are $\mathbf{R}_{face}$ and $\mathbf{p}_{face}$, respectively.

Joystick Control. The task is to match the character's center-of-mass (COM) velocity with a 2D planar target velocity that is parallel to the ground (see Figure 1b). The reward function we use is $r_t = exp(-5\|\frac{\mathbf{v}_{com}}{\|\mathbf{v}_{com}\|} - \frac{\mathbf{v}_{target}}{\|\mathbf{v}_{target}\|}\|^2) \cdot exp(-5\|\|\mathbf{v}_{com}\| - \|\mathbf{v}_{target}\|\|^2)$, where $\mathbf{v}_{com}$ and $\mathbf{v}_{target}$ are the character's COM and the target velocities, respectively. The goal state $\mathbf{g}_t = \mathbf{R}_{face}^{-1}(\mathbf{v}_{target} - \mathbf{v}_{com})$ includes the difference of the two velocities that is represented with respect to the current facing transformation.

Path Follower. The task is to follow the given ∞-shaped path on the ground (see Figure 1c). The goal point at time t is given by $\mathbf{x}_t = A(sin(bt), sin(bt)cos(bt))$, where we used 10 and 2 for A and b, respectively. The reward function is $r_t = exp(-\|\mathbf{p}_{com} - \mathbf{x}_t\|^2) \cdot exp(-2.5\|\mathbf{v}_{com} - \dot{\mathbf{x}}_t\|^2)$, where we measure the position and velocity differences between the COM and the current goal. The goal state includes both the current and the two future goal points $(\mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \dot{\mathbf{x}}_t, \dot{\mathbf{x}}_{t+1}, \dot{\mathbf{x}}_{t+2})$, which are converted into values

relative to the current facing transformation (similar to the previous two tasks).

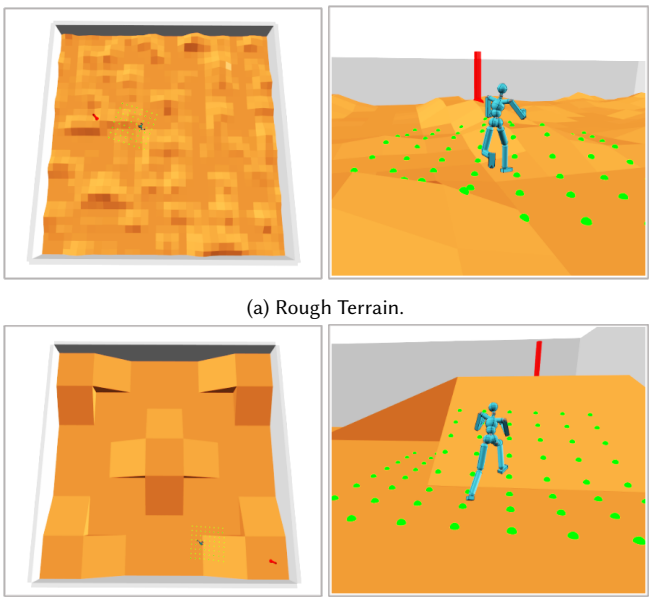
Maze Runner. Given a maze completely surrounded by walls, the goal is to explore the maze as fully as possible in a fixed time (120 seconds). Figure 1d shows the maze used in our experiment, where we discretize the entire map (24m by 24m) into a 8x8 grid to compute rewards efficiently, the agent gets a sparse reward of 1 when entering a new region, then that region is marked as visited. Once the agent cover the entire map, it gets a sparse reward of 30, then the episode terminates. As a result, the maximum return (the sum of rewards in an episode) is 94 (64+30). For the goal state, we use vision input and visit information in the vicinity of the character. More specifically, the vision input includes two consecutive visual sensory inputs $(\mathbf{l}_{t-1}, \mathbf{l}_t)$ where each input $\mathbf{l}_t = (d_1, d_2, \cdots, d_{32})$ includes 32 rays that are 5 meters long maximum and span 270 degree wide, where d_i is the distance of i-th ray which stops when it collides other objects. The visit information is a 3x3 bit matrix marking whether the regions around the character position have been previously visited or not. The matrix corresponds to 5m by 5m in the maze.

The deep RL control policies built by our method were able to complete all the tasks successfully while also generating natural-looking motions. In *Point-Goal Navigation* and *Joystick Control*, our simulated characters were very agile when the goal position or the target velocity were changed, respectively. Although the path that we used in *Path Follower* was more challenging because it is composed of sharper turns, our controller showed very accurate path following capability. In *Maze Runner*, our character was able to visit all the cells within 90 seconds. Please note that hierarchical RL was used to disentangle high-level navigation planning and low-level motion planning in the previous study [Ling et al. 2020], however, our control policy (non-hierarchical) was able to solve the task directly. This result implies that our pre-trained controller has the capability of the low-level motion planning that is mostly related to maintaining balance. Consequently, deep RL algorithms can learn high-level navigation planning directly.

5.3 Other Downstream Tasks

In addition to the baseline downstream tasks, we also test *Uneven Terrains* to understand how our control policy structure with the helper branch adapts to unseen environments and *Crowd Simulation* to understand how effective the pre-trained locomotion controllers are in solving multi-agent environments. We use the helper branch with $\sigma = (1.0, 1.0, \cdots)$ for better adaption to complex scenarios.

5.3.1 Uneven Terrains. *Uneven Terrains* tasks are basically the same as the *Point-Goal Navigation* task except that the flat ground is replaced with challenging uneven terrains. We test on a rough terrain made by adding a random offset into the height map (maximum 1 meter) and a terrain that includes five peaks of 4 meters where the character must climb and descend approximately 38 degree inclines to transit between the peaks (see Figure 5). To enable the character to observe the terrains, a local height map in the vicinity of the current position of the character is included in the goal state. More specifically, we use a 3m by 3m patch discretized into 8 by 8 cells where the character is located slightly behind the center to be able to see more vision to the front and less to the back (see the green dots in Figure 5).



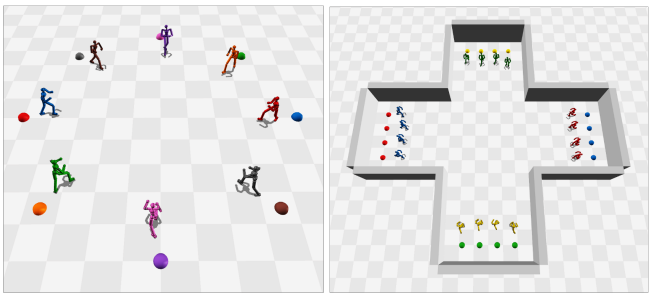
(a) Rough Terrain.

(b) Five Peaks Terrain.

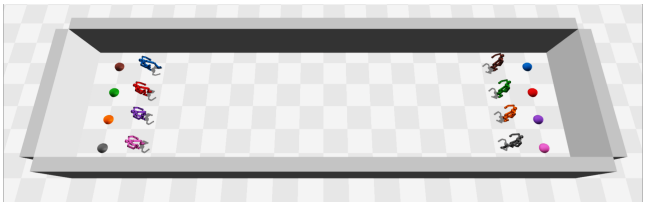
Fig. 5. Two examples of rough terrain. The top (a) was made by adding a random offset to the height map and the bottom (b) was manually designed. The green dots represent a local height map near the simulated character.

Our character is able to perform the navigation task successfully in both terrains, it is also resilient to unexpected external perturbations and can recover from perturbed states. The character often detoured around steep areas to reduce the risk of falling down, which is similar to what humans do. These examples show the main benefit of using physically simulated characters, which is the adaptation to new environments, whereas the behaviors generated by kinematics-based controllers are strictly limited by what is included in the input motion capture data.

5.3.2 Crowd Simulation. We hypothesize that the robustness of our pre-trained controllers will be especially helpful in tasks that involve complex interaction with other agents. To validate our hypothesis we run crowd simulations using physically simulated humanoids for three environments, *Circle*, *Carrefour*, and *Corridor*, where each agent is asked to walk to a designated goal position (see Figure 6). Crowd simulation using physically simulated humanoids was first demonstrated in [Haworth et al. 2020], where a hierarchical policy structure with a shared value function was used. In contrast to their method, we use a very simple formulation with no hierarchical control policy and no multi-agent specific deep RL algorithm. The goal state includes visual sensory inputs (rays) to observe the other agents and the relative goal position with respect to the agent's



(a) Circle (b) Carrefour



(c) Corridor

Fig. 6. Crowd Simulation Environments

facing transformation. The reward function is

$$r_t = r_{\text{goal}} + r_{\text{reached}} - 0.1\, r_{\text{collision}} \tag{3}$$

$$r_{\text{goal}} = d_{t-1} - d_t$$

$$r_{\text{reached}} = \begin{cases} 1 \text{ if the agent is close to the goal position} \\ 0 \text{ otherwise} \end{cases}$$

$$r_{\text{collision}} = \begin{cases} 1 \text{ if collision occurs with any other agent} \\ 0 \text{ otherwise} \end{cases}$$

where d_t is the distance to the designated goal position. The first term brings the agents closer to the goal position, the second term encourages them to stay close after arrival, and the third term penalizes collisions among the agents.

In *Circle* and *Carrefour*, the agents showed a behavior that could be observed in a typical traffic circle. In *Corridor*, the agents swerved to the right to prevent collisions. In our experiment setting, successful control policies (i.e., each agent reaching their own goal position) were learned within one day for *Circle* and *Corridor*, two days for *Carrefour*. The difference in learning speed is reasonable because the *Carrefour* example includes 16 agents whereas the others include 8 agents, so more collisions could happen at the center location and the physics simulation also takes more time.

5.4 Other Datasets

To test the generality of our method, we test with other datasets: a set of boxing motions selected from the CMU motion capture dataset [CMU 2002], and a set of dancing motions from the AIST dataset [Tsuchida et al. 2019]. Unlike the locomotion dataset, the boxing and dancing motions do not contain enough transitions among the different behaviors in each dataset. This characteristic can lead to the *sinking* problem as pointed out in [Ling et al. 2020], where the learned controller generates only a small subset of behaviors and does not transit to the other behaviors regardless of what random latent vectors are given. To solve this problem, we construct a motion graph using a loose connectivity threshold to allow many transitions between the different behaviors. We then generate motions that are 10 minutes long by traversing the nodes of the graph in a random order, and we follow the procedure described in section 5.1 to learn the controllers. Figure 7 shows random simulation rollouts generated by our task-agnostic controllers for the two datasets. All the characters start with the same state at the beginning (the left most screenshot), and gradually follow different paths in the space over time (the right most screenshot).

We develop a downstream task that is suitable for the boxing dataset, which we call *Boxing Bag Training* (see Figure 8). The goal is to hit the boxing bag as hard as possible with the gloves while avoiding collisions with the other body parts. The boxing bag weighs 40 kg and has a spherical joint on its top so that it can move when hit by the boxer character. The goal state $\mathbf{g}_t = (\mathbf{p}_t, \mathbf{v}_t, \mathbf{h}_t)$ involves the relative position $\mathbf{p}_t$ and linear velocity $\mathbf{v}_t$ of the bag with respect to the character's root joint and a 2-bit array $\mathbf{h}_t$ representing the expected pattern of punches. During learning, we randomly generate patterns among left-punch $(1, 0)$, right-punch $(0, 1)$, and no-punch $(0, 0)$. The reward function is

$$r_t = 0.2 * r_{\text{close}} + r_{\text{hit}} - 0.1\, r_{\text{collision}} \tag{4}$$

$$r_{\text{close}} = exp(-5\|d_t - 1.2\|^2)$$

$$r_{\text{hit}} = \big(max(f_R - 100, 0), max(f_L - 100, 0)\big) \cdot \mathbf{h}_t \tag{5}$$

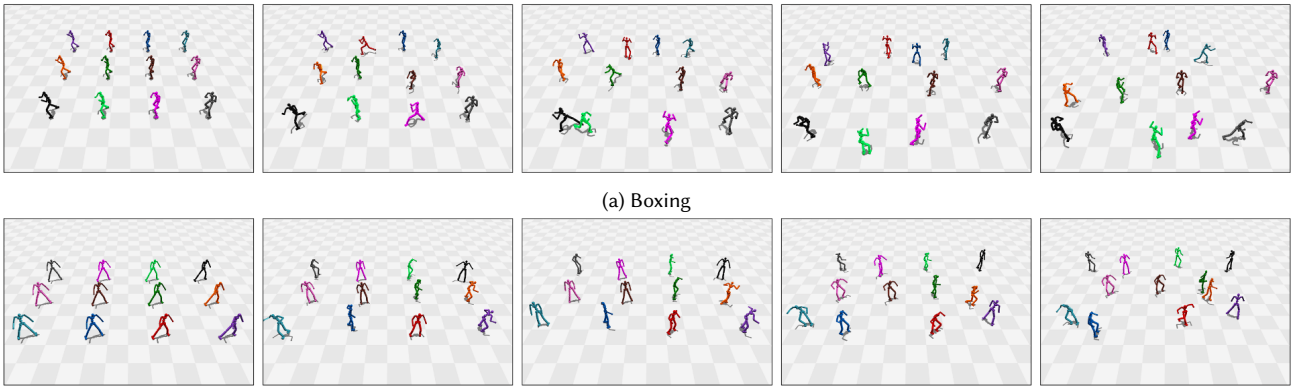
$$r_{\text{collision}} = \begin{cases} 1 \text{ if unexpected collision occurs} \\ 0 \text{ otherwise} \end{cases}$$

where d_t is the distance measured on the ground plane between the bag and the boxer and f_R, f_L are contact forces measured in the right and left gloves, respectively. The first term r_{close} encourages the boxer to keep an appropriate distance (1.2m in our setting) from the boxing bag, the second term r_{hit} becomes positive only when the boxer hits the boxing bag by using the glove that matches to the expected punch pattern and the contact force occurred is larger than 100N, and the third term $r_{\text{collision}}$ penalizes any collision that does not match the current punch pattern. Learning a plausible control policy takes approximately 1 day (200M tuples), and then the boxer can hit the bag reliably with the punch patterns changing over time.

5.5 Evaluation

We compare our full model with other models in the literature and observe the differences quantitatively and qualitatively.

5.5.1 Rollout Performance. Our conditional VAE has two major technical components: the physics simulation layer and the stochastic latent variables. To understand how the components affect the results, we compare performance with various settings. More specifically, we measure performance by the elapsed time until the simulated character falls down (*time-until-fall*). First, we change the weight on the simulation consistency α, where zero is equivalent to using no simulation layer during training. Figure 9 shows the average and the standard deviation over 100 simulation rollouts, where we terminate a rollout either if the character falls down or if



(a) Boxing

(b) Dancing

Fig. 7. Random rollouts generated by the controllers learned with boxing and dancing datasets. The screenshots are shown in chronological order.

Fig. 8. *Boxing Bag Training* experiment setup

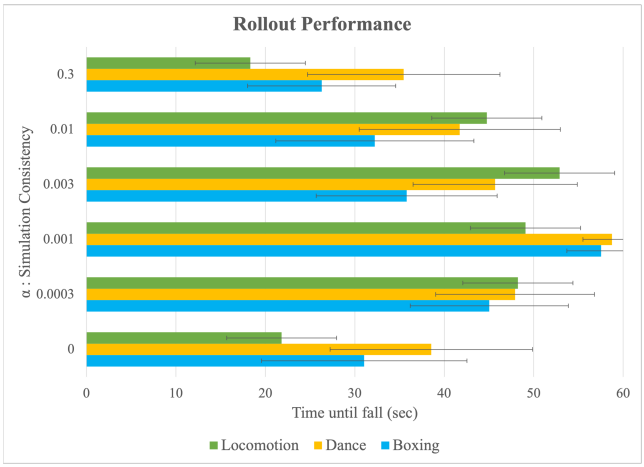


Fig. 9. Rollout performance comparison for various α (simulation consistency).

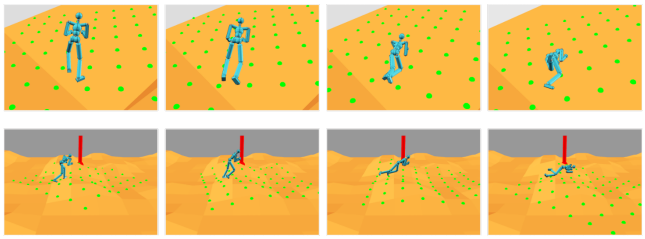


Fig. 10. Screenshots of the deep RL control policies without helper branches.

the length exceeds 60 seconds. For every dataset, the models trained with the simulation layer consistently performed three times better, where the best performances were shown in the values between 0.0003 to 0.003. If the weight is too high, the performance was degraded because the behavior cloning loss could be dominated by the simulation consistency. Second, we learned controllers without the stochastic latent variables (i.e., no KL term in the loss), the characters fell within less than 1 second (please refer to the supplemental video). Those results make sense because non generative models can generate meaningful outputs only when conditioned by the outputs of their original encoders.

5.5.2 Helper Branch. We do an ablation study on the helper branch to understand its effectiveness when learning downstream tasks that are not fully covered by the input expert trajectories. *Point-Goal Navigation* tasks on uneven terrains serve as a good example because the locomotion dataset only includes motions captured on flat ground. Figure 10 shows results when the deep RL control policies without a helper branch (Figure 3a) are used for uneven terrain tasks. The characters can walk a few steps on a relatively flat surface, however, they failed on rougher terrain. On the other hand, the characters could perform both tasks successfully when used with the policy that includes the helper branch (see Figure 5) demonstrating that the helper branch provided effective adaptation for such challenging tasks.

We run an experiment to understand the effectiveness of the adaptation parameter σ (Figure 3b). This parameter allows users to

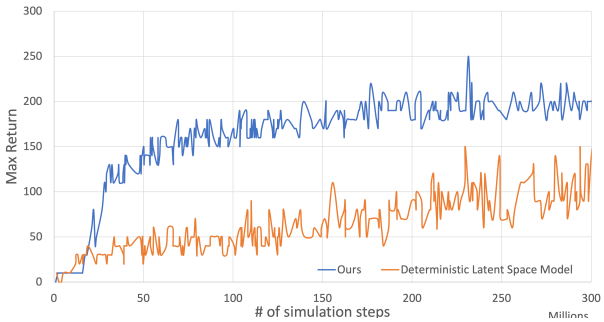
control style when the input expert trajectories include all the motor skills required to perform a downstream task. When this is the case, there are many solutions and the user can control the style of the solution. Using a small value provides better style preservation but task performance could be degraded. A large value provides the opposite behavior. We show how visually distinctive results could be created with different adaptation parameters by comparing parameter values of 0.1 and 0.5 for the *Boxing Bag Training* task. Because the radian is used to measure angles in our experiment setup, values of 0.1 and 0.5 mean that the deep RL control policies can alter the output (joint angles) of the pre-trained controller up to approximately 5.7 and 28.6 degrees, respectively. In both experiments, we achieve plausible control policies, however, the results were visually distinctive. The control policy learned with the higher parameter value could generate contact forces up to 500N while showing energetic movements, whereas the maximum contact force generated by the policy learned with the lower parameter value was approximately 200N and generated motions that looked more similar to the original motions.

5.5.3 Performance Comparison to Other Latent Space Models in Downstream Tasks with Sparse Rewards. In environments composed of high-dimensional state and action spaces such as ours, it is very challenging to learn plausible control policies when only sparse rewards are available. Deep RL algorithms get little meaningful learning signals (i.e., feedback) from the environment and there could be many local minima (i.e., under-constrained). As a result, carefully engineered dense rewards are typically required to learn plausible control policies for physically simulated humanoids, however, the reward engineering is non-trivial and time-consuming.

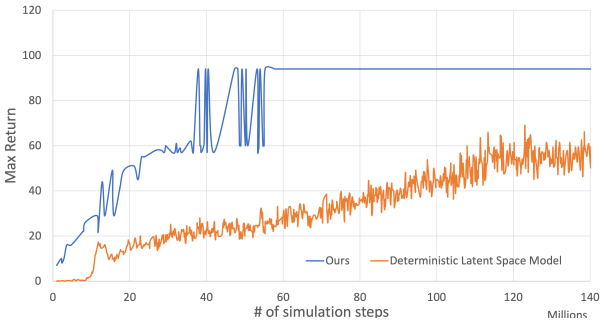
It would be ideal if plausible control policies could be learned successfully from simple and intuitive sparse rewards. To understand the benefits of using our conditional VAE model in learning with sparse rewards, we compare the performance between our control policy structure and an alternative structure, a latent space model that resembles to the structure used in [Merel et al. 2019a, 2020; Won et al. 2021]. Note that we report a latent space model without the *AR(1)* prior because it did not produce a meaningful improvement in our experiments. We compare two different control policy structures by using the *Point-Goal Navigation* task with the reward function modified as follows:

$$r_t = \begin{cases} 1 \text{ if } d_t \text{ is less than 0.5m} \\ 0 \text{ otherwise} \end{cases}$$

where d_t is the distance to the goal point from the character's root position projected on the ground. Figure 11a shows the maximum return achieved during learning, where our model performs better than the alternative latent space model due to significantly better exploration capability especially in the initial learning phase. A qualitative comparison is also demonstrated in the supplemental video, where the motions generated by our model look natural and resemble the original motions in the locomotion dataset. We also run another experiment with the *Maze Runner* task, which requires a more complex strategy and already uses a sparse reward (i.e., the character gets 1 only when entering new unvisited area). Figure 11b



(a) Point-Goal Navigation (Sparse)



(b) Maze Runner

Fig. 11. Performance Comparison in the Sparse Reward Environments.

illustrates the maximum return achieved during the learning, where the graph shows that our control policy structure is able to learn much faster and eventually cover the entire map within 60M training tuples whereas the alternative model can not cover the entire map even after generating 140M tuples.

6 CONCLUSION

In this paper, we develop controllers for physically simulated humanoid characters using conditional VAEs that can generate various motions by conditioning the controllers on a random vector sampled from the standard Gaussian distribution. We also demonstrate various use cases for the pre-trained controllers by learning deep RL control policies for various downstream tasks.

Although the capability of pre-trained controllers can be extended by the control policy structure using a helper branch during the downstream task learning, the base capability is still limited by the input expert trajectories just as with other data-driven methods. The base capability can be considered from two perspectives, the motion quality and the variability. First, if the input expert trajectories have flaws in their motion quality, our controller also generates motions with the same flaws by matching the style. For example, in the early stages of development, we mistakenly used reference motions with heights taller than our simulated characters, which generated a stomping-gait because the simulated character tried to match the reference root joint position. Second, our controller can also have the *sinking* problem mentioned in [Ling et al. 2020], where it fails to transition among different behaviors if the input trajectories lack such transitions. Using datasets composed of many heterogeneous

behaviors where individual recordings are relatively short might cause this problem. Augmenting datasets by constructing motion graphs, which we used in our experiments, might be a quick remedy, however, preparing datasets rich in transitions would create the most natural-looking motions.

We tested our model with the CMU dataset [CMU 2002] included in the AMASS dataset [Mahmood et al. 2019] to further understand how scalable our conditional VAE model is. Because there are many short clips less than 5 seconds in the dataset, we also construct a motion graph and generate 2 hours of motion clips as described in section 5.4. From this experiment, we observed that the model learned from the CMU dataset does not perform well when compared to other models learned from either the locomotion dataset or the dancing dataset. We assume that the diversity of the dataset is beyond our model capacity because the current single motor-decoder structure might not be suitable for learning many heterogeneous behaviors simultaneously as pointed out in [Won et al. 2020]. Learning task-agnostic physics-based controllers such as our conditional VAE model with large uncurated datasets such as the CMU or the entire AMASS dataset still remains an open problem.

In our current implementation, we built the physics simulation layer separately based on supervised learning with input expert trajectories. Due to approximation errors, the learned simulation layer could generate slightly different output from the PyBullet physics simulator. Recently, many differential physics simulators have been proposed [Heiden et al. 2021; Nimble 2021]. Using one of these, we could unify all the simulation-related components (the simulation layer and the actual simulator) as a single neural physics simulator. Such unification would provide outputs that are always consistent with actual simulation rollouts when learning controllers, so ideal performance could be achieved.

One exciting future direction would be to learn the controllers directly from a motion capture dataset without going through expert trajectories generated by imitation policies. Because the actions $\mathbf{a}_t$ are not available to access with motion capture data only, the behavior cloning loss could be omitted in Equation 1 by incorporating a neural physics simulator. Another promising direction would be to apply the controllers to more complex multi-agent competitive tasks such as [Liu et al. 2021], where 2:2 soccer game strategies for physically simulated agents are demonstrated. Because our method provides an efficient way of learning downstream tasks while preserving the original motion styles well, we believe that more realistic and natural-looking soccer strategies could be achieved.

REFERENCES

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-driven Responsive Control of Physics-based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (2019). http://doi.acm.org/10.1145/3355089.3356536

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

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

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust Task-based Control Policies for Physics-based Characters. *ACM Trans. Graph.* 28, 5, Article 170 (2009). http://doi.acm.org/10.1145/1618452.1618516

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Trans. Graph.* 29, 4, Article 130 (2010). http://doi.acm.org/10.1145/1778765.1781156

Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. 2011. Locomotion Skills for Simulated Quadrupeds. *ACM Trans. Graph.* 30, 4 (2011). https://doi.org/10.1145/2010324.1964954

Erwin Coumans and Yunfei Bai. 2016–2019. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Levi Fussell, Kevin Bergamin, and Daniel Holden. 2021. SuperTrack: Motion Tracking for Physically Simulated Characters using Supervised Learning. *ACM Trans. Graph.* 40, 6, Article 197 (2021). https://dl.acm.org/doi/10.1145/3478513.3480527

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-based Locomotion for Bipedal Creatures. *ACM Trans. Graph.* 32, 6, Article 206 (2013). http://doi.acm.org/10.1145/2508363.2508399

S. Ghorbani, C. Wloka, A. Etemad, M. A. Brubaker, and N. F. Troje. 2020. Probabilistic Character Motion Synthesis Using a Hierarchical Deep Latent Variable Model. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. https://doi.org/10.1111/cgf.14116

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust Motion In-Betweening. 39, 4 (2020).

Leonard Hasenclever, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel. 2020. CoMic: Complementary Task Learning amp; Mimicry for Reusable Skills. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 4105–4115. https://proceedings.mlr.press/v119/hasenclever20a.html

Brandon Haworth, Glen Berseth, Seonghyeon Moon, Petros Faloutsos, and Mubbasir Kapadia. 2020. Deep Integration of Physical Humanoid Control and Crowd Navigation. In *Motion, Interaction and Games (MIG '20)*. Article 15. https://doi.org/10.1145/3424636.3426894

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. 2021. NeuralSim: Augmenting Differentiable Simulators with Neural Networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. https://github.com/google-research/tiny-differentiable-simulator

Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. 2020. MoGlow: Probabilistic and Controllable Motion Synthesis Using Normalising Flows. *ACM Trans. Graph.* 39, 6, Article 236 (2020), 14 pages. https://doi.org/10.1145/3414685.3417836

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. 1995. Animating Human Athletics. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. 71–78. https://doi.org/10.1145/218380.218414

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (2017). http://doi.acm.org/10.1145/3072959.3073663

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 (jul 2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. 1996. Limit Cycle Control and Its Application to the Animation of Balancing and Walking. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. 155–162. https://doi.org/10.1145/237170.237231

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-objective Control. *ACM Trans. Graph.* 37, 6, Article 180 (2018). http://doi.acm.org/10.1145/3272127.3275071

Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. 2021. Learning a Family of Motor Skills from a Single Motion Clip. *ACM Trans. Graph.* 40, 4, Article 93 (2021). https://doi.org/10.1145/3450626.3459774

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). http://doi.acm.org/10.1145/3306346.3322972

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

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

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. arXiv:1712.09381

Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel Van De Panne. 2020. Character Controllers Using Motion VAEs. *ACM Trans. Graph.* 39, 4, Article 40 (2020). https://doi.org/10.1145/3386569.3392422

Michael L. Littman. 1994. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning (ICML'94)*. 157–163.

Libin Liu and Jessica Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Trans. Graph.* 36, 3, Article 42a (2017). http://doi.acm.org/10.1145/3083723

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). http://doi.acm.org/10.1145/3197517.3201315

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). http://doi.acm.org/10.1145/2893476

Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, S. M. Ali Eslami, Daniel Hennes, Wojciech M. Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, Noah Y. Siegel, Leonard Hasenclever, Luke Marris, Saran Tunyasuvunakool, H. Francis Song, Markus Wulfmeier, Paul Muller, Tuomas Haarnoja, Brendan D. Tracey, Karl Tuyls, Thore Graepel, and Nicolas Heess. 2021. From Motor Control to Team Play in Simulated Humanoid Football. arXiv:2105.12196

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. 2020. CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion. *ACM Trans. Graph.* 39, 4 (2020), 10 pages.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. 2019. AMASS: Archive of Motion Capture as Surface Shapes. In *The IEEE International Conference on Computer Vision (ICCV)*. https://amass.is.tue.mpg.de

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. 2019a. Hierarchical Visuomotor Control of Humanoids. In *7th International Conference on Learning Representations, ICLR 2019*. https://openreview.net/forum?id=BJfYvo09Y7

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. 2019b. Neural Probabilistic Motor Primitives for Humanoid Control. In *7th International Conference on Learning Representations, ICLR 2019*. https://openreview.net/forum?id=BJl6TjRcY7

Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. 2020. Catch Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks. *ACM Trans. Graph.* 39, 4, Article 39 (2020). https://doi.org/10.1145/3386569.3392474

Nimble. 2021. Nimble Physics. https://nimblephysics.org/.

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 (2019). http://doi.acm.org/10.1145/3355089.3356501

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.

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). http://doi.acm.org/10.1145/3197517.3201311

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.* 34, 4 (2015), 80:1–80:11. http://doi.acm.org/10.1145/2766910

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). http://doi.acm.org/10.1145/3072959.3073602

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. 2019. MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. In *Advances in Neural Information Processing Systems 32*. 3681–3692.

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

Hoseok Ryu, Minseok Kim, Seunghwan Lee, Moon Seok Park, Kyoung-Min Lee, and Jehee Lee. 2020. Functionality-Driven Musculature Retargeting. *Computer Graphics Forum* 40 (2020), 341–356.

Kihyuk Sohn, Xinchen Yan, and Honglak Lee. 2015. Learning Structured Output Representation Using Deep Conditional Generative Models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15)*. 3483–3491.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural state machine for character-scene interactions. *ACM Trans. Graph.* 38, 6 (2019), 209:1–209:14. https://doi.org/10.1145/3355089.3356505

Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Trans. Graph.* 39, 4, Article 54 (2020). https://doi.org/10.1145/3386569.3392450

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011a. Articulated Swimming Creatures. *ACM Trans. Graph.* 30, 4 (2011). https://doi.org/10.1145/2010324.1964953

Jie Tan, C. Karen Liu, and Greg Turk. 2011b. Stable Proportional-Derivative Controllers. *IEEE Computer Graphics and Applications* 31, 4 (2011), 34–44. https://doi.org/10.1109/MCG.2011.30

Shuhei Tsuchida, Satoru Fukayama, Masahiro Hamasaki, and Masataka Goto. 2019. AIST Dance Video Database: Multi-genre, Multi-dancer, and Multi-camera Database for Dance Information Processing. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019*. Delft, Netherlands, 501–510.

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). http://doi.acm.org/10.1145/2185520.2185521

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. 2020. DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

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

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2021. Control Strategies for Physically Simulated Characters Performing Two-Player Competitive Sports. *ACM Trans. Graph.* 40, 4, Article 146 (2021). https://doi.org/10.1145/3450626.3459761

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

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 (2017).

Jia-chi Wu and Zoran Popović. 2003. Realistic modeling of bird flight animations. *ACM Trans. Graph.* 22, 3 (2003).

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

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). http://doi.acm.org/10.1145/1778765.1778811

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

Zhiqi Yin, Zeshi Yang, Michiel Van De Panne, and Kangkang Yin. 2021. Discovering Diverse Athletic Jumping Strategies. *ACM Trans. Graph.* 40, 4, Article 91 (2021). https://doi.org/10.1145/3450626.3459817