A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters

JUNGDAM WON, Facebook AI Research
DEEPAK GOPINATH, Facebook AI Research
JESSICA HODGINS, Facebook AI Research

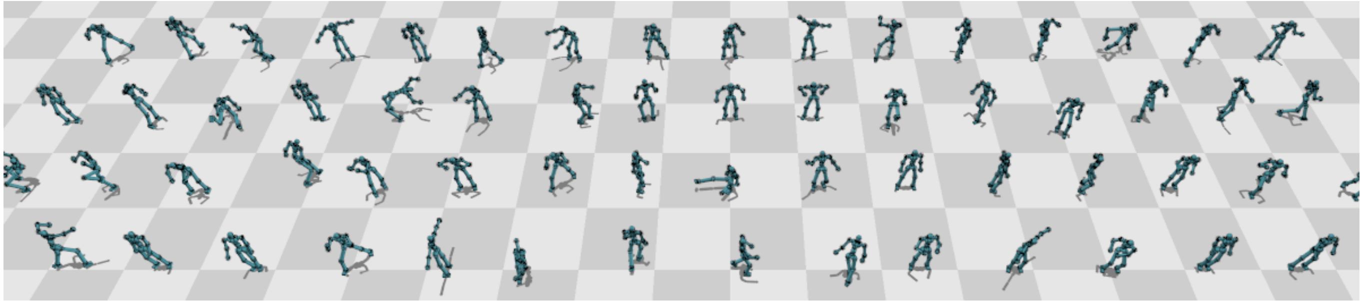


Fig. 1. Using a database with many, heterogeneous motion clips, our framework learns a single dynamic controller that generates a large variety of motions.

Human characters with a broad range of natural looking and physically realistic behaviors will enable the construction of compelling interactive experiences. In this paper, we develop a technique for learning controllers for a large set of heterogeneous behaviors. By dividing a reference library of motion into clusters of like motions, we are able to construct *experts*, learned controllers that can reproduce a simulated version of the motions in that cluster. These experts are then combined via a second learning phase, into a general controller with the capability to reproduce any motion in the reference library. We demonstrate the power of this approach by learning the motions produced by a motion graph constructed from eight hours of motion capture data and containing a diverse set of behaviors such as dancing (ballroom and breakdancing), Karate moves, gesturing, walking, and running.

CCS Concepts: • **Computing methodologies** → **Animation**; *Physical simulation*; *Reinforcement learning*; *Neural networks.*

Additional Key Words and Phrases: Character Animation, Physics-based Simulation and Control, Reinforcement Learning, Deep Learning, Neural Network, Locomotion Control

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

1 INTRODUCTION

Human characters that can move and behave naturally in real-time are required if we are to create compelling populated virtual worlds. Because human behaviors are governed by physical laws, we can ensure physical realism in human motion and interactions with objects and the environment by incorporating the laws of physics into the motion generation process through simulation. However, physics is not a sufficient constraint to guarantee the naturalness of human behaviors. Learning natural control policies for physically simulated humanoid characters is challenging because the characters are under-actuated (there are more degrees of freedom (DOF) than controllable DOFs) and because there are many physically correct ways to perform a task or behavior that do not appear natural (for example, requiring unrealistically high torques or employing strategies that a human would not use).

Recently, imitation learning approaches that provide an effective way to generate natural looking motions for physically simulated humanoid characters have been developed. The core idea is to learn a control policy that allows a simulated character to successfully imitate selected motion capture clips. With recent advances in deep reinforcement learning, these approaches are able to reproduce realistic and diverse behaviors in a physically realistic and natural looking manner (see, for example, [Bergamin et al. 2019; Liu and Hodgins 2017, 2018; Park et al. 2019; Peng et al. 2018]). However, they are still limited in their ability to scale to large, heterogeneous datasets. Current approaches only learn from a relatively small corpus of homogeneous data, where each learned policy is only effective on a limited set of behaviors. Combining together multiple learned behaviors to create a character with a broad repertoire of behaviors is not a solved problem.

In this paper, we propose a control framework for physically simulated humanoid characters that can scale to model many hours of human motion data and generate motions that look natural. The input of our framework is a motion controller, which can be a motion graph [Kovar et al. 2002; Lee et al. 2002], phase-functioned neural network (PFNN) [Holden et al. 2017], or any other motion generation technique (kinematic or dynamic, data-driven or simulated). The output is a universal control policy (dynamic controller) that can

Authors' addresses: Jungdam Won, Facebook AI Research; Deepak Gopinath, Facebook AI Research; Jessica Hodgins, Facebook AI Research.

simulate all the motions generated from the input controller. More precisely, the output control policy is a dynamic controller that produces control inputs (e.g., target posture or joint torques) for the simulated character that allow it to perform the task. First, we generate a large library of reference motions from the input motion controller and cluster the motions to group like motions together. Second, we build dynamic controllers, which we call *experts*. These are specialized controllers trained using deep reinforcement learning to produce the behaviors contained in a single cluster. Finally, we combine the experts together and learn a single controller by using a *mixture-of-experts* method in order to produce the entire library of reference motions. Our framework is general enough to handle several different input controller types and can generate a large variety of different motions from a single combined controller.

To show the power of our approach, we demonstrate a dynamic controller learned for a large motion graph constructed from eight hours of motion capture data containing a wide variety of different behaviors. These experiments demonstrate the ability of our approach to scale both to larger data sets and to a variety of behaviors. The behaviors includes not only ordinary behaviors such as walk, run, standing but also unusual behaviors such as crawling, acting, breakdancing, Latin dance, and Indian dance. A second dynamic controller learned from data produced by a PFNN demonstrates that the framework is agnostic to the input controller. For this second controller, we show real-time generation of the simulated motion by using a joystick. Our approach based on motion clustering allowed us to learn both controllers successfully, while learning from scratch failed. We also use a suite of experiments to explore the important design choices and improvements in the RL controllers used in our approach.

The primary contribution of this paper is a framework for creating controllers that handle a diverse and heterogeneous set of human motions. The dataset used to train the controller is significantly larger than that typically used in prior research on techniques for physically based characters, we focus on generating motions that are physically correct and natural. Additional contributions include:

- **Learning Experts based on Motion Clusters.** To make the learning of a general controller feasible, we first divide the dataset into clusters of like motion and train *expert* controllers on those clusters. The experts are then combined with a gating function to train a combined controller. This hierarchical approach makes the problem of learning a general control tractable.
- **Enhancement to RL-based Imitation Learning.** We found a good combination of existing techniques for RL-based imitation learning, which shows good performance and eliminates a need for behavior-specific manual tuning.
- **Baseline for Future Learned Controllers** We learned a general controller for a large dataset comprised of many different behaviors, so it is possible that our controller can serve as a warm start for other control system designs in much the way that models trained on *ImageNet* have been used to bootstrap many different classification and labeling problems in computer vision.

2 RELATED WORK

We review the most closely related literature in deep reinforcement and imitation learning for character control. Our approach is based on combining a number of controllers, so we review ensemble methods including mixture-of-experts approaches that have been used in other domains. We also explore prior work that forms the foundation for our approach, including kinematic controllers, closed loop control, open loop control, muscle-based control, and model-based control. Because one of the advantages of our approach is the ability to handle a wide variety of behaviors, we also look at work that has controlled multiple and unusual behaviors.

2.1 Physics-based Character Control

Physics-based control is a popular approach to generating human motion, because it ensures physical realism in motions of the character and interactions with the environment. Many different approaches have been proposed: open-loop control via optimization [Liu et al. 2010; Mordatch et al. 2012; Sok et al. 2007] and closed-loop feedback control [Coros et al. 2010; da Silva et al. 2017; Lee et al. 2010; Liu et al. 2016; Mordatch and Todorov 2014; Ye and Liu 2010; Yin et al. 2007]. Model-based control is an another popular approach, where simplified physical models such as an inverted pendulum are used to predict the current and future trajectory and controls are performed based on that information [da Silva et al. 2008; Hämäläinen et al. 2015; Kwon and Hodgins 2010, 2017; Macchietto et al. 2009; Mordatch et al. 2010; Tassa et al. 2012]. Although the characters are typically actuated by joint torques (motors), some studies developed controllers for muscle-actuated characters to make the characters move in a natural manner [Geijtenbeek et al. 2013; Lee et al. 2019, 2009, 2014; Nakada et al. 2018; Wang et al. 2012].

2.2 Deep Reinforcement Learning

Since Peng and his colleagues [2016] introduced deep reinforcement learning (Deep RL) for physics-based characters to the animation research community, these approaches have gained in popularity due to their simple formulation and powerful control capability. Deep RL controllers for various characters have been proposed for quadrupeds [Peng et al. 2016; Yu et al. 2018], flying creatures [Won et al. 2017, 2018], deformable underwater creatures [Min et al. 2019], and humanoids [Berseth et al. 2018; Clegg et al. 2018; Lee et al. 2019; Liu and Hodgins 2017, 2018; Merel et al. 2017; Peng et al. 2018; Wang et al. 2017; Won and Lee 2019; Yu et al. 2018]. For humanoid characters, controller design based on imitation learning produces natural looking results when the reward is computed by comparing the given motion capture data (reference motion) and the state of the simulated character. However, these approaches are still limited in their ability to scale to large, heterogeneous datasets and the algorithms only converge when used with a relatively small corpus of data (i.e. each learned policy is only effective on a limited set of behaviors). Controllers for a large number of motion capture clips have also been developed. Bergamin et al. [2019] demonstrated a low-cost and robust controller for Motion Matching which is suitable for real-time applications such as games. Park et al. [2019] demonstrated a controller that is tightly coupled with RNN kinematic controller. Both methods were able to handle 1-2 hours of motion capture

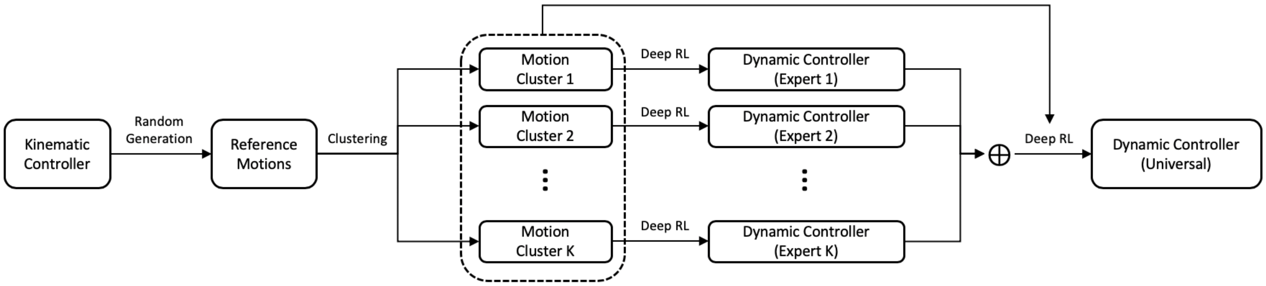


Fig. 2. Overview of the system for learning a dynamic controller for a large set of behaviors.

clips that are composed of homogenous behaviors. Chentanez et al. [2018] proposed a Deep RL formulation with PD controllers, the gains of which are changed by Deep RL policy on the fly. Merel et al. [2019b] proposed an architecture called Neural Probabilistic Motor Primitives for policy transfer of individually learned policies. Both methods were tested on the complete CMU database. However, the generated motions still have motion-quality limitations, and avoid challenging motions, such as breakdancing. In this paper, we overcome these problems by learning a mixture of expert controllers, each of which handles a class of motions.

2.3 Ensemble Methods

The ensemble method is in common use in the machine learning community. The basic idea is to combine several primitive components (e.g., basis functions, learned features, experts on specific task) into a high level component to solve bigger problems. The primitives can be learned in a hierarchical manner [Frans et al. 2017; Hausknecht and Stone 2016; Merel et al. 2019a,b], or an end-to-end manner [Bacon et al. 2017; Sutton et al. 1999; Vezhnevets et al. 2017]. Our controller is based on a hierarchical approach, which uses a mixture-of-experts, each expert provides effective control over only part of the desired space of behaviors. We use a weighted sum to combine actions from the experts [Jacobs et al. 1991] to achieve control over the whole space. Recently, Peng et al. [2019] proposed a multiplication method to combine the experts, however, we opted for the summation instead because it showed better performance in our experiments. In physics-based character animation research, there exist earlier studies which focused on composing different controllers [da Silva et al. 2009; Faloutsos et al. 2001], hierarchical methods have also been used to solve high-level tasks [Coros et al. 2009; Peng et al. 2019] and locomotion skills [Peng et al. 2016, 2017].

2.4 Kinematic Controllers

The input of our framework is a controller that is used to generate a large reference library and we use existing kinematic controllers for this task. Many kinematic controllers have been developed on top of the motion graph construct [Kovar et al. 2002; Lee et al. 2002], which builds a graph by treating each pose as a node and providing connections when poses are sufficiently similar. In the game industry, Motion Matching [Clavet 2016] is often used. This approach is similar to a motion graph but optimized for real time performance and responsiveness. Recently, controllers based on deep neural networks (DNN) have been developed, DNN with varying weights [Holden et al. 2017; Starke et al. 2019; Zhang et al. 2018], RNN [Lee et al. 2018]. To show the generality of our approach, we built two systems, one uses a motion graph constructed from the entire CMU motion capture database, and the other one uses a phase-functioned neural network (PFNN) motion controller [Holden et al. 2017].

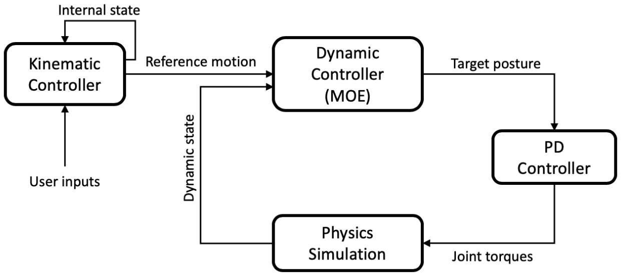


Fig. 3. Overview of the run-time system

3 OVERVIEW

Our framework takes a kinematic controller capable of generating a broad variety of behaviors as input and creates a universal dynamic controller that can generate physically simulated motion that is similar to that of the input controller. Figure 2 illustrates the framework. The first step is to create a library of reference motions. We do this by generating a large number of long (10 s) sequences by applying random inputs to a controller such as a motion graph or PFNN. The goal is to have a library that is large enough to cover the range of behaviors that that controller can produce. The reference library is then clustered to group like motions together. Motions from each cluster are used to train dynamic controllers using deep reinforcement learning (Deep RL). We call those controllers *experts*. The experts are combined together, and a dynamic controller is trained for the entire reference library by using a *mixture-of-experts* (MOE) method.

Figure 3 depicts the run-time system that generates a new motion. The input kinematic controller is used to create a new reference motion that should be simulated, based on a random walk or user input. The dynamic controller (MOE) generates a target posture for the simulated character while considering the physical state of the character and the reference motion. The character is actuated by joint torques computed from PD servos, and the physical simulation updates the state of character.

We explain each of these steps in more detail in the next three sections.

4 REFERENCE MOTION PREPARATION

Given the input kinematic controller, we first create reference motions from the controller, then cluster them into groups of like motions. The clustered reference motions will be inputs to the learned dynamic controller. The framework is agnostic to the form of the input controller and we test with two: a large motion graph constructed from eight hours of data representing a variety of different behaviors, and an available PFNN model for locomotion [Holden et al. 2017].

4.1 Motion Graph Construction

We use the CMU motion capture dataset [CMU 2002] included in the AMASS dataset [Mahmood et al. 2019] to build a motion graph to serve as one of our input controllers. It has more than nine hours of motion capture, spanning over 100 subjects. The motions in the dataset are highly diverse and unstructured because the data were captured from different subjects and marker setups. We excluded motions that contained physical interactions with other subjects, large or heavy objects, or rough terrain in a semi-automatic manner because motions that include significant interaction with the environment are not realizable in our framework. Approximately eight hours of motion remained. The AMASS dataset uses the statistical body shape prior called *SMPL+H* [Loper et al. 2015], where the body shape and motions are modeled as separate parameters. As a result, motions can be retargeted by assigning the same body shape parameters to all motions. We used a male model whose body shape parameters are that of the average male. The root joint is offset based on the minimum height for the contact points in each motion. This simple algorithm does not fix all the contact errors caused by retargeting but remaining errors will be fixed through the dynamic controller.

We then construct a motion graph [Kovar et al. 2002; Lee et al. 2002] from the retargeted motions. The motion graph is represented as a directed graph structure $\mathbb{G} =< \mathbb{N}, \mathbb{E} >$, where $\mathbb{N}, \mathbb{E}$ are a set of nodes, edges, respectively. A node represents one temporal moment, for which we use a frame number n by assuming a fixed FPS (frames per second). The existence of an edge from frame n to m implies that playing frame m after frame n will not result in a visible discontinuity in the motion. Given motion clips with a total of K frames, we first create K nodes, then compare each node pair and create an edge from frame n to frame m if the distance $D(n, m)$ is less than the user-provided threshold δ. We consider several consecutive differences simultaneously to enforce smoother transitions.

$$\begin{gathered} D(n,m) = \sum_{k \in \kappa} d(n+k, m+k) \\ d(n,m) = w_1 \frac{d_{\mathrm{q}}}{|\mathcal{J}|} + w_2 \frac{d_{\mathrm{v}}}{|\mathcal{J}|} + w_3 \frac{d_{\mathrm{ee}}}{|\mathcal{F}|} + w_4 d_{\mathrm{root}} \end{gathered} \tag{1}$$

where κ is a temporal window, for which we use $\{0.0, 0.3, 0.6\}$ seconds. $d(n, m)$ is per-frame difference from frame n to frame m that includes four terms, d_{q}, d_{v}, d_{ee}, and d_{root} are joint angle, joint velocity, end-effector positions, and the root joint position differences, respectively, w_i is a relative weight for each difference. $\mathcal{J}, \mathcal{F}$ are sets of all joints, end-effectors, whose cardinalities are $|\mathcal{J}|$, $|\mathcal{F}|$. Normalizing each difference makes tuning the weight values more intuitive. The four terms are defined as follows

$$\begin{aligned} d_{\mathrm{q}} &= \sum_{j \in \mathcal{J}} \alpha_j \| \log((\mathbf{q}_j^n)^{-1} \cdot \mathbf{q}_j^m) \|^2 \\ d_{\mathrm{v}} &= \sum_{j \in \mathcal{J}} \alpha_j \| \omega_j^n - \omega_j^m \|^2 \\ d_{\mathrm{ee}} &= \sum_{j \in \mathcal{E}} \frac{exp\big(-(h_j^n)^2/\sigma_h\big)}{\sum_{j \in \mathcal{E}} exp\big(-(h_j^n)^2/\sigma_h\big)} \| \mathbf{p}_j^n - \mathbf{p}_j^m \|^2 \\ d_{\mathrm{root}} &= \| \bar{\mathbf{p}}^n - \bar{\mathbf{p}}^m \|^2 \end{aligned} \tag{2}$$

where $\mathbf{q}_j \in \mathrm{SO}(3)$, $\omega_j \in \mathbb{R}^3$ are the orientation and angular velocity of j-th joint (both values are represented with respect to the parent joint), α_j is a joint weight value. We use quaternions to represent orientations, and log is the logarithm of quaternion which maps the quaternions to an axis-angle representation. $\mathbf{p}_j \in \mathbb{R}^3$ is a position of j-th joint, h_j is a height from the ground, and $\bar{\mathbf{p}} \in \mathbb{R}^2$ is a projection of the root joint position on the ground. The term d_{ee} is not only encouraging a match between end-effector positions but also inferring contact states of the end-effectors automatically from their height above the ground. End-effectors that are close to the ground have a large weight value. This fully automatic inference of contact states is advantageous for large motion datasets composed of highly diverse behaviors because manual or semi-automatic contact state labeling is impractical. Once edge generation is complete, we find the largest weakly connected component from the graph to prune dangling nodes. We experimented with using just the strongly connected component as was done by [Lee et al. 2002], but too many nodes were pruned from the graph because of the diversity of the behaviors in our graph.

4.2 Motion Generation and Clustering

We generate reference motions from the motion graph $\mathbb{G} =< \mathbb{N}, \mathbb{E} >$. The motion is generated by starting from a randomly selected node n, traversing to an adjacent node m through a random outgoing edge. The traversal process repeats until the generated motion is a given length L (10 s in our implementation). Because our graph is a weakly connected component, a random walk in the graph could arrive at a dead end, where the current node has no outgoing edge. When this happens, we backtrace until an unvisited outgoing edge appears, and repeat the random traversal. If the backtrace reaches the initial start node, we choose another start node randomly. We created 8192 reference motions, approximately 23 hours in total.

Motion clustering is used to divide the reference motion library into groups that contain homogeneous data. The clustering information will be utilized to train the experts on groups of like behaviors and to ensure that the training data for the universal dynamic controller is balanced across the kinds of behaviors in our reference library. We developed a simple but efficient clustering algorithm inspired by [Onuma et al. 2008]. We first compute a feature vector $\mathbf{c} = (K_1^{\parallel}, K_1^{\perp}, E_1, \cdots, K_{|\mathcal{J}|}^{\parallel}, K_{|\mathcal{J}|}^{\perp}, E_{|\mathcal{J}|})$ for each motion

$$K^{\parallel} = \frac{1}{N}\sum_{j}^{N} \|\mathbf{v}_j^{\parallel}\|^2,\ K^{\perp} = \frac{1}{N}\sum_{j}^{N} \|\mathbf{v}_j^{\perp}\|^2,\ E = \frac{1}{N}\sum_{j}^{N} \|\mathbf{a}_j\|^2 \qquad (3)$$

where N is the number of frames, $K^{\parallel}, K^{\perp} \in \mathbb{R}$ are average kinetic energies in planar and vertical directions, respectively, $E \in \mathbb{R}$ is an approximation of energy expenditure, $\mathbf{v}_j^{\parallel} \in \mathbb{R}^2$ and $\mathbf{v}_j^{\perp} \in \mathbb{R}$ are linear velocities projected to the ground and to the gravitational direction, $\mathbf{a}_j \in \mathbb{R}^3$ is the acceleration of the j-th joint. Once the feature vectors are computed, we run a *k-means* clustering algorithm to create 8 clusters for our experiments.

5 DYNAMIC CONTROLLER LEARNING

We learn a combined dynamic controller for a set of *experts*, which are dynamic controllers learned individually for each cluster of reference motions. The dynamic controller is a universal control policy that can produce a physically correct version of any reference motion generated from the input controller. This hierarchical learning strategy allows the control to learn a large database of diverse reference motions more efficiently. For our dynamic controllers, we use non-linear policies represented by deep neural networks, which are learned by an imitation learning method similar to that used by [Peng et al. 2018]. We made several improvements so that it can be applied to a large dataset including diverse and heterogeneous behaviors. These are described below.

5.1 Reinforcement Learning Formulation

Reinforcement learning is a method that can be used to control a Markov Decision Process, where the agent interacts with the environment by performing an action $\mathbf{a} \in \mathcal{A}$ determined by its policy $\pi : \mathcal{S} \longrightarrow \mathcal{A}$, where $\mathcal{S}$, $\mathcal{A}$ are domains of states, actions, respectively, a state $\mathbf{s} \in \mathcal{S}$ is an observation of the environment from the agent. Whenever a new action $\mathbf{a}$ is performed, the current state $\mathbf{s}$ changes into the new state $\mathbf{s}'$, and the agent gets a reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ which provides information about how desirable the state transition was. The goal of reinforcement learning is to find the optimal policy that maximizes the average cumulative rewards $\sum_{i=0}^{\infty} \gamma^i r_i$, where $\gamma \in (0, 1)$ is a discount factor that determines how far into the future is considered by the policy and also makes the sum finite.

The state in our RL formulation has the state of the physically simulated character, the states from reference motions, and the state differences between the simulated character and the reference motions:

$$\mathbf{s} = (\mathbf{s}_{\text{sim}}, \mathbf{s}_{\text{ref}}^1, \cdots, \mathbf{s}_{\text{ref}}^{\text{K}}, \mathbf{s}_{\text{ref}}^1 \ominus \mathbf{s}_{\text{sim}}, \cdots, \mathbf{s}_{\text{ref}}^{\text{K}} \ominus \mathbf{s}_{\text{sim}}) \qquad (4)$$

Similar to previous work [Peng et al. 2018], the state of the physically simulated character $\mathbf{s}_{\text{sim}} = \{\mathbf{p}_j, \mathbf{q}_j, \mathbf{v}_j, \omega_j\}_{j=1:|\mathcal{J}|}$ includes the position $\mathbf{p}_j \in \mathbb{R}^3$, orientation $\mathbf{q}_j \in \mathbf{SO(3)}$, linear velocity $\mathbf{v}_j \in \mathbb{R}^3$, and angular velocity $\omega_j \in \mathbb{R}^3$, where we assume the index 1 is the root joint. All values are represented with respect to the facing transformation $(\bar{\mathbf{q}}, \bar{\mathbf{p}})$, which is a projection of the root joint transformation to the ground.

$$\begin{aligned} \bar{\mathbf{p}} &= \mathbf{p}_1 - \frac{\mathbf{p}_1 \cdot \mathbf{u}_{\text{up}}}{\|\mathbf{u}_{\text{up}}\|^2} \\ \bar{\mathbf{q}} &= \text{RotationMatrixToQuaternion}\big((\mathbf{x}^T, \mathbf{y}^T, \mathbf{z}^T)\big) \\ \mathbf{x} = \mathbf{y} \times \mathbf{z}, \quad \mathbf{y} &= \frac{\mathbf{v}_{\text{up}}}{\|\mathbf{v}_{\text{up}}\|}, \quad \mathbf{z} = \frac{\bar{\mathbf{z}}}{\|\bar{\mathbf{z}}\|} \\ \bar{\mathbf{z}} &= \mathbf{u}_{\text{facing}} - \frac{\mathbf{u}_{\text{facing}} \cdot \mathbf{u}_{\text{up}}}{\mathbf{u}_{\text{up}} \cdot \mathbf{u}_{\text{up}}} \end{aligned} \qquad (5)$$

where $\bar{\mathbf{q}}, \bar{\mathbf{p}}$ are the orientation, position of the root joint projected to the ground (Figure 5), $\mathbf{u}_{\text{up}}$ is the up-vector direction that is perpendicular to the ground and parallel to the gravity direction, $\mathbf{u}_{\text{facing}}$ is the facing direction of the root joint. The input reference motions are represented by $(\mathbf{s}_{\text{ref}}^1, \cdots, \mathbf{s}_{\text{ref}}^{\text{K}})$, where we use three sample points 0.3, 0.6, and 0.9 seconds ahead from the current time t. We compute each sample point $\mathbf{s}_{\text{ref}}^i$ in the same manner as $\mathbf{s}_{\text{sim}}$. The state differences between the simulated character and reference motions $(\mathbf{s}_{\text{ref}}^1 \ominus \mathbf{s}_{\text{sim}}, \cdots, \mathbf{s}_{\text{ref}}^{\text{K}} \ominus \mathbf{s}_{\text{sim}})$ are included to increase performance and to accelerate learning speed [Bergamin et al. 2019]. Please note that we use information that can be created automatically from the reference motions (e.g., joint angles and velocities). The design is similar to [Park et al. 2019], except for the addition of the state differences as outlined above. This state representation is advantageous in learning a controller for a large heterogeneous dataset that includes highly diverse and unstructured behaviors because it does not require any prior knowledge and manual labels on input reference motions such as a phase variable [Peng et al. 2018] or the internal state of kinematic controllers [Bergamin et al. 2019]. Furthermore, it enables transfer learning between dynamic controllers, given that all the dynamic controllers share the same state representation. The action space of our dynamic controller is a target posture $(\mathbf{q}_2, \mathbf{q}_3, \cdots, \mathbf{q}_{|\mathcal{J}|})$ excluding the root joint.

Our reward function provides a way to measure the difference between the simulated character and the current reference motion. The reward function is a multiplication of five terms

$$r = r_{\text{pose}} \cdot r_{\text{vel}} \cdot r_{\text{ee}} \cdot r_{\text{com}} \cdot r_{\text{root}} \qquad (6)$$

where r_{pose}, r_{vel} measure differences of joint angles, velocities, respectively. The terms r_{ee} and r_{com} reward matches in the positions of the end-effectors and center-of-mass and are the same as those used in [Peng et al. 2018]. Because all terms are computed from relative values measured with respect to the root joint, they only measure relative differences. To reduce global differences between the simulated character and the reference motion, we add a final term r_{root}

$$r_{\text{root}} = exp\Big(-\frac{w_q\|\log(\bar{\mathbf{q}}_{\text{sim}}^{-1} \cdot \bar{\mathbf{q}}_{\text{ref}})\|^2 + w_p\|\bar{\mathbf{p}}_{\text{sim}} - \bar{\mathbf{p}}_{\text{ref}}\|^2}{\sigma_{\text{root}}}\Big) \qquad (7)$$

where $(\bar{\mathbf{q}}_{\text{sim}}, \bar{\mathbf{p}}_{\text{sim}})$, $(\bar{\mathbf{q}}_{\text{ref}}, \bar{\mathbf{p}}_{\text{ref}})$ are facing transformations as explained in equation 5, w_q, w_p are relative weights for orientations, positions, respectively, and σ_{root} is a degree of sensitivity on the difference.

The beauty of reinforcement learning is that it can learn complex control policies from a simple scalar-valued reward function. However, because of that simplicity, the definition of the reward function can have a significant impact on the resulting control policy. We tested several reward function designs that had been used in prior work for reference motions ranging from simple ones (standing, walking, and running) to complex ones (ballroom dancing, breakdancing, and gymnastics). We found that a multiplicative reward [Lee et al. 2019; Park et al. 2019] has several benefits in imitation learning compared to the additive reward used in [Peng et al. 2018, 2017]. First, it shows better performance for reference motions that require agile and subtle behaviors such as dancing. In our experiments, the additive reward usually satisfied only a few terms rather than balancing the reward among all terms. With the multiplicative reward, all terms were forced to be non-zero to obtain a reasonable reward. The multiplicative reward also has fewer parameters because weights to balance among the terms are not needed because the standard deviations in the exponential functions only influence the importance of each term, whereas the relative weights existing outside the exponential functions also influence in the additive reward. We also found that different behaviors require different relative weights when the additive reward function is used but not when the reward is multiplicative. We show the effectiveness of the multiplicative reward over the additive reward in Section 6

5.2 Learning

We use the proximal policy optimization (PPO) algorithm [Schulman et al. 2017] to learn the dynamic controller. The algorithm utilizes stochastic policy gradients with the generalized advantage estimator $GAE(\lambda)$ and multi-step returns $TD(\lambda)$. An important idea of PPO is to prevent the new policy from changing significantly from the current policy by using a clipped surrogate objective with a clip parameter ϵ. The algorithm has been used in many of the state-of-the-art examples of physics-based character control [Bergamin et al. 2019; Lee et al. 2019; Park et al. 2019; Peng et al. 2018; Won and Lee 2019; Yu et al. 2018].

Early termination ends the current episode before the predefined maximum length when the motion has failed. The most popular termination method is to check collisions between the simulated character and the ground. For example in an walking controller, we can check whether any body is in contact with the ground other than the feet. As pointed out in [Peng et al. 2018; Won and Lee 2019], controller learning was not possible without early termination because it increases sample efficiency significantly and prevents the algorithms from falling into bad local minima.

In our system, simple early termination conditions are hard to identify because our reference motions are diverse. [Bergamin et al. 2019] proposed checking the height of the head but we have motions on which the character spins on his head. Further, our reference motions include motions in which the points of contact change such as walking followed by crawling. Instead of using behavior-specific early termination conditions, we perform early termination based on the reward value similar to [Babadi et al. 2019; Xie et al. 2019]. More precisely, we terminate the current episode if the average reward value for l s is less than $\rho \times r_{max}$, where l and ρ are temporal and qualitative tolerances, respectively, and r_{max} is the maximum value of the reward function, which is 1.0 in Equation 6. We found that early termination by reward best performs when it is used with the multiplicative reward because the reward value is equally affected by all sub-terms and terminates episodes whenever the current episode does not satisfy any sub-term, this increases sample efficiency.

Table 1. Simulation and learning parameters

Simulation	Simulation rate (Hz)	480
	Control rate (Hz)	30
Learning	Policy learning rate (α_π)	$1.0e^{-5}$
	Value learning rate (α_V)	$1.0e^{-3}$
	Discount factor (γ)	0.95
	GAE and TD (λ)	0.95
	# of tuples per policy update (T)	8192
	Batch size for policy update (n)	256
	Iteration for policy update (N)	10
	Clip parameter (ϵ)	0.2

5.3 Learning a Mixture-of-Experts Policy

Learning a single controller for heterogeneous behaviors is challenging because strategies must be learned simultaneously to control different behaviors such as walking, crawling, breakdancing, and ballroom dancing. To address this challenge, we propose a controller based on mixture-of-experts (MOE). Figure 4 shows an example of the MOE structure. Given the reference motions clustered into eight groups, we first learn dynamic controllers $\pi_0, \cdots, \pi_7$ (experts) separately. The experts are specialized controllers for each cluster, and they are combined together with a gating network. The combined controller $\pi(\mathbf{a}|\mathbf{s})$ is computed by a weighted sum of the experts.

$$\pi(\mathbf{a}|\mathbf{s}) = \sum_{i \in \mathcal{E}} w_i(\mathbf{s})\pi_i(\mathbf{a}|\mathbf{s}), \;\; w_i(\mathbf{s}) = \frac{exp(g_i(\mathbf{s}))}{\sum_{i \in \mathcal{E}} exp(g_i(\mathbf{s}))} \tag{8}$$

where $\mathcal{E}$ is a set of the experts, $w_i(\mathbf{s})$ is a weight on the i-th controller whose value is computed by the softmax values on the output of the gating network $\mathbf{g}(\mathbf{s}) = (g_1(\mathbf{s}), g_2(\mathbf{s}), \cdots) \in \mathbb{R}^{|\mathcal{E}|}$ Finally, we learn the combined controller with the entire library of reference motions. During learning, all weights for the gate network and the experts are updated end-to-end to allow further improvement for the entire library. We re-sampled reference motions so that the number of motions in each cluster are similar. This resampling reduces data imbalance that leads to the *forgetting* problem in learning a combined controller, where experts easily lose their specializations and adapt only to new data. For example, if we learn a controller by random selection from the database, all expert controllers would be adapted to the ordinary behaviors such as walking because they appear most frequently in the database. This hierarchical strategy has several benefits compared to learning from scratch. First, the expert controllers can be tuned to focus on smaller and homogeneous data, which allows us to cover broader range of behaviors. Second, the structure is scalable and reusable. When new reference motions are added, existing experts can be combined with a new expert to make another controller.

6 RESULTS

Figure 5 shows our simulated characters and Table 2 includes their physical properties. We have different models for the characters

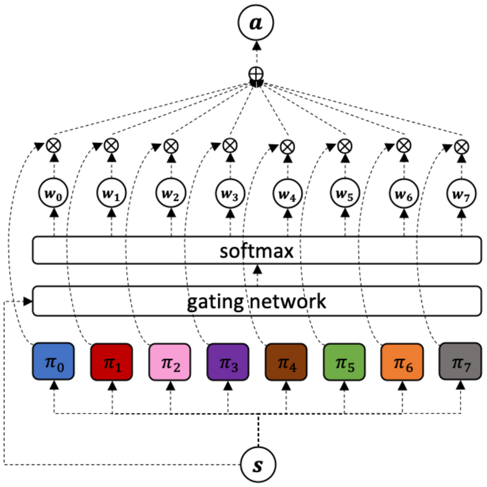
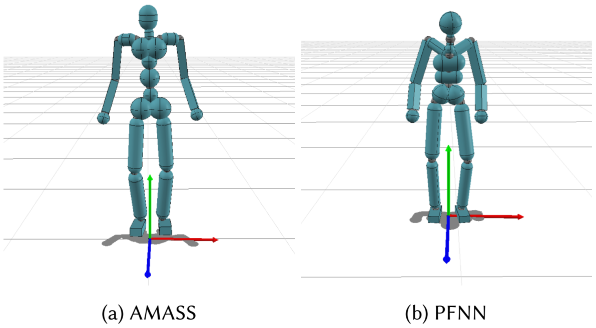


Fig. 4. A Mixture-of-Experts Structure. An output action **a** is generated by a weighted sum of the outputs of the eight experts $\pi_1 \cdots \pi_8$, where the gating network followed by the softmax function generates the weights.


(a) AMASS

(b) PFNN

Fig. 5. Characters

Table 2. Properties of the characters

Property	**AMASS**	**PFNN**
Joints (=Links)	19	22
Weight (kg)	53.5	48.3
Height (m)	1.61	1.50
Degrees of freedom	57	60
State dimension	1880	2153
Action dimension	51	54
Lateral Friction Coefficient	0.8	
Rolling Friction Coefficient	0.3	
Restitution Coefficient	0.0	

depending on the input controller (motion graph or PFNN) because we use the provided model for PFNN with no changes [Holden et al. 2017]. Our characters are modeled as articulated rigid bodies, where bodies are connected by 3-DOF spherical joints and 1-DOF revolute joints. The hierarchies of our characters match the reference motions except that we removed the finger and toe joints. We adopt similar mass distributions to [Peng et al. 2018], and attached a stable PD servo [Tan et al. 2011] with manually assigned gains and torque limits for each joint.

We used PyBullet [Coumans and Bai 2019] to model and simulate the physical environment and the OpenAI baseline [Dhariwal et al. 2017] implementation for the PPO algorithm. All expert controllers use neural networks with two fully-connected layers and 256 hidden units for each layer. The gating network in the combined controller uses a neural network with two fully-connected layers and 128 hidden units in all experiments, where the internal layers use ReLu non-linear activation. Table 1 includes the parameter values used for simulation and Deep RL. The same values were used for all experiments unless noted.

All computations were performed on 2 CPUs (Intel Xeon CPU E5-2698 v4), with 40 physical cores total. Building a motion graph, generating reference motions, and clustering takes approximately 6, 1, and 0.5 hours, respectively. The learning of the expert controller usually converges after 1e8 transition tuples are generated, which takes approximately 3 days in our implementation. For clusters that contain difficult motions such as breakdancing, training requires twice as much time. Learning a combined controller for the entire reference library requires 3e8 tuples for full convergence (approximately 10 days).

To generate 1 s of motion at run-time, requires approximately 1 s when a motion graph is used to produce the new reference motion (the motion that will be simulated) and 1.5 s when a PFNN controller is used (with a single thread). The time includes the generation of the new reference motion, evaluation of the deep neural network controller, and the physics simulation. PFNN requires more time to generate the new reference motion because of the network evaluation. Our implementation for all components is in Python and presumably a C/C++ implementation would provide better run-time performance.

6.1 Motion Clustering

We clustered the entire reference library of 8192 motion clips of 10 s each into eight groups. The left image in Figure 6 shows a two-dimensional view of the clustering results and representative postures corresponding to the clusters are shown on the right. The number of motions varies across the cluster. For example, the largest cluster (cluster 0) includes approximately 4000 motions, the smallest cluster (cluster 7) only includes 8 motions. This imbalance is because our motion database has many ordinary behaviors such as walking, running, and standing, while it has only a few clips for unusual behaviors such as breakdance, Indian dance, and ballroom dance.

6.2 Mixture-of-Experts Controller

As described in Section 5.3, we sampled reference motions equally from each cluster when learning the combined controller to prevent the controller from only learning the motions in the large clusters. Our controller was able to generate most of the reference motions successfully, it can also generate motions from reference motions

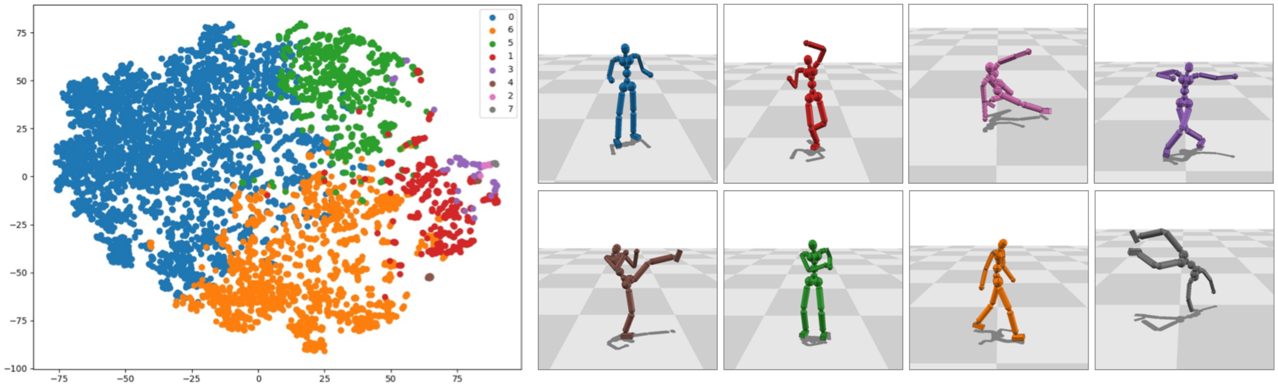


Fig. 6. Motion Clustering Results. Two-dimensional embedding by t-SNE (Left) and representative postures corresponding to the clusters (Right).

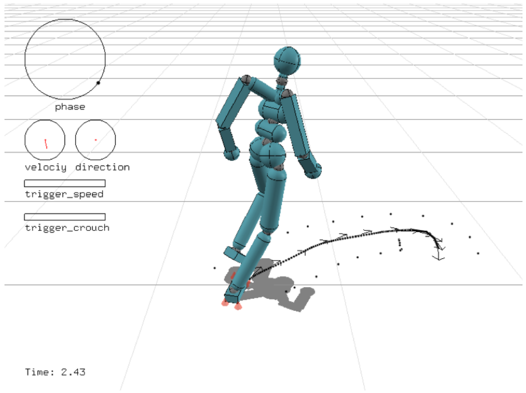


Fig. 7. Dynamic Controller for PFNN

which were not seen during training. Figure 1 depicts a set of simulated characters imitating reference motions (seen and unseen). All of the characters are controlled by the single combined controller.

6.3 Comparing Controller Designs

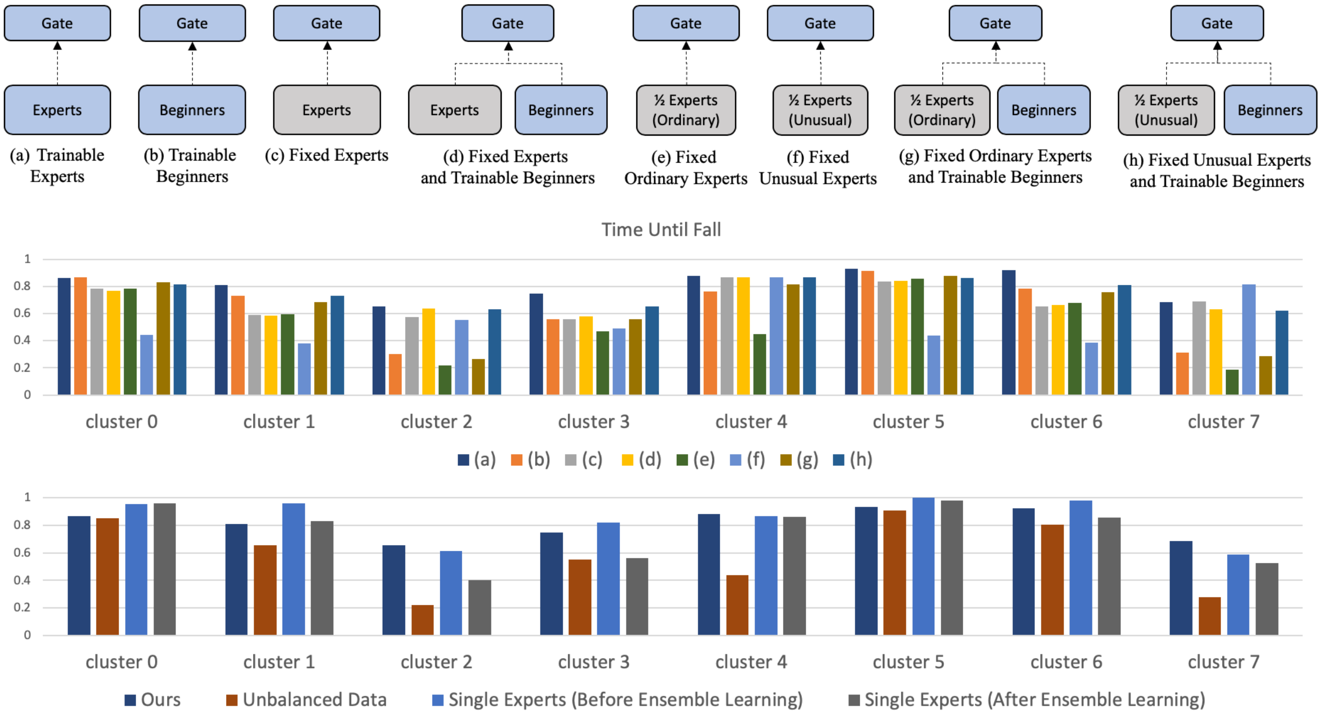
To understand the effectiveness of the design of our controller, we compared its performance with other possible design choices (see Figure 8). We explore the importance of experts in controller design and how best to reuse existing experts when new motions are added. A beginner is a controller whose structure is the same as an expert but it is not pre-trained for any specific cluster (i.e. the network weights are randomly initialized). Blue indicates that the controller is trainable during the training, whereas grey means that the weights are fixed. We experimented with eight different designs: (a) our MOE controller (experts are all pre-trained, cover all eight clusters and can be modified during the training), (b) all beginners (learning from scratch), (c) the experts are fixed during combined controller learning, (d) mix of fixed experts and beginners (e) fixed experts trained for only ordinary behaviors such as walking, running are included (f) fixed experts trained for unusual behaviors such as dancing, (g) an addition of beginners to (e), and (h) addition of beginners to (f).

The second graph in Figure 8 shows a comparison of those eight designs. We measured performance of the controllers by the average elapsed time until falling K. Given a reference motion with length T s, K means that the character falls down after $K \times T$ s. For example, $K = 0.5$ implies that the character falls down after half the time has passed. We randomly selected 100 motions from each cluster to compute the values. The result shows that our controller (a) is superior to other controllers for all clusters on average. We spent the same training time for all designs. Mixture of beginners (b) does not learn the difficult motions (clusters 2, 3, 7) successfully. As expected the fixed experts in (c) perform less well than trainable experts (a). We had hoped that beginners (d) would improve the performance over fixed experts but that was not the case. It appears that training beginners and the gate function at the same time is too difficult. The final four designs (e-h) are additional experiments with beginners (with either the ordinary or unusual experts deleted) and again the beginners fail to compensate for the missing experts.

The third graph in Figure 8 illustrates the importance of training on balanced data for the clusters that have few motions (2, 4, and 7) where the performance is significantly degraded when training occurs with a randomly selected (and therefore unbalanced) data set. The last two bars in this figure shows that the performance of experts on the cluster that they were originally trained on is lower after they are trained in concert with the mixture graph. This result implies that there is a trade-off between generalization (the ability to create a combined controller that can perform all motions in a large corpus) and specialization (the ability to perform just one behavior well).

6.4 Additive vs. Multiplicative Reward

Figure 9 shows a comparison between dynamic controllers learned by the additive reward (top) and the multiplicative reward (bottom). All weights and variances for the reward functions were kept fixed for all experiments. In a simple and low-energy reference motion (a), we see only a small difference in the pose, but our multiplicative controller overshoots a little less. In the reference motion (b), the foot positions are more accurate with the multiplicative controller. In one of the most challenging reference motions, (c), which includes agile footsteps, a back-flip, and a handstand, only the multiplicative



(a) Trainable Experts (b) Trainable Beginners (c) Fixed Experts (d) Fixed Experts and Trainable Beginners (e) Fixed Ordinary Experts (f) Fixed Unusual Experts (g) Fixed Ordinary Experts and Trainable Beginners (h) Fixed Unusual Experts and Trainable Beginners

Fig. 8. Performance Comparison. Design choices for combined controllers (Top) and their performances (Bottom)

controller was successful at learning the behavior even after the additive controller trained for 2e8 transition tuples. These results imply that our multiplicative reward is general enough to be applied to highly diverse behaviors without requiring behavior-specific parameter tuning.

6.5 Early Termination based on the Reward

We compare our early termination strategy based on reward values to one of the most popular strategies, which is based on checking collisions with the ground. Figure 10 shows the qualitative comparisons. For collision-based early termination, we need to manually select the body parts that are not allowed to collide with the ground. Which body parts should be selected depends on the behavior. In contrast, our reward strategy does not require any behavior-specific prior knowledge. Our early termination strategy was able to learn dynamic controllers for both reference motions as was the manually determined termination strategy.

6.6 Application to Other Kinematic Controllers

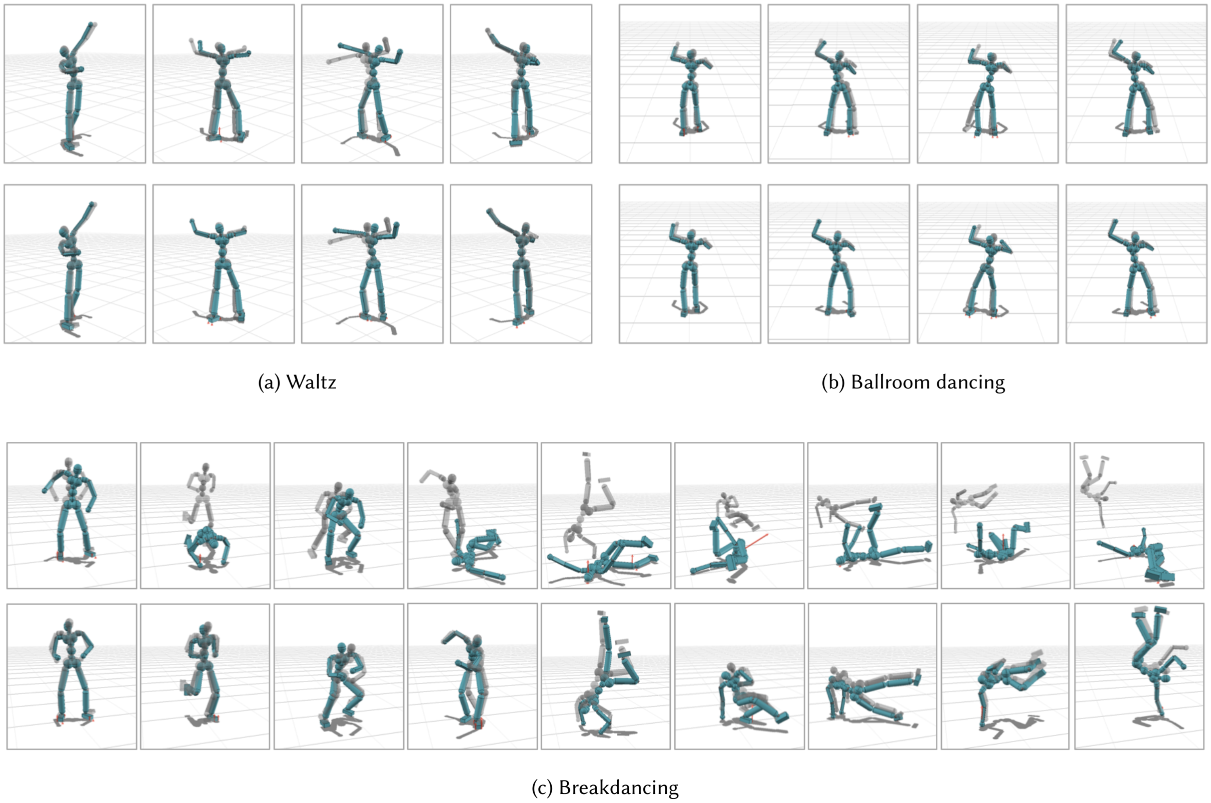
Most kinematic controllers can easily be integrated into our system as illustrated in Figure 2. To show the generality of our system, we also built a system using an existing state-of-the-art kinematic controller, phase-functioned neural networks (PFNN) [Holden et al. 2017], as the input controller. It requires continuous user inputs to generate motions, where there exist directional (analog stick) and discrete (button) inputs. To mimic the directional input automatically, we use a momentum-based random walk inspired by [Peng et al. 2016].

$$\begin{aligned} u_i &= \mathrm{clip}(u_{i-1} + \Delta u \cdot m_i, -1, 1) \\ m_i &= \mathrm{clip}(m_{i-1} + \Delta m_i, -m_{\max}, m_{\max}) \\ \Delta m_i &= \mathrm{sign}(U(-1,1) - \frac{m_{i-1}}{m_{\max}})) \cdot U(0, \Delta m_{\max}) \end{aligned} \tag{9}$$

where U is an uniform distribution for given ranges, u_i, m_i are the current user input, its momentum, respectively. Δu, $m_{\max}$, and $\Delta m_{\max}$ control the speed of input change, where 0.3, 0.5, and 0.1 were used in our experiment. To mimic the discrete input, we randomly trigger the input with the probability ρ, keep it turned on for a random duration between $(\delta_{\min}, \delta_{\max})$, and repeat this with a fixed interval ν. We used $\rho = 0.1$, $\delta_{\min} = 1.0$, $\delta_{\max} = 5.0$, and $\nu = 1.0$. Because the PFNN controller was learned for locomotion only, two hours of reference motions with three clusters were enough for learning the dynamic controllers in our experiment, where the clusters approximately correspond to standing, crouching, and a mix of walking and running, respectively. Figure 7 shows a screenshot where the simulated character is successfully controlled interactively using joystick inputs from a human user. Examples are included in the accompanying video.

7 CONCLUSION

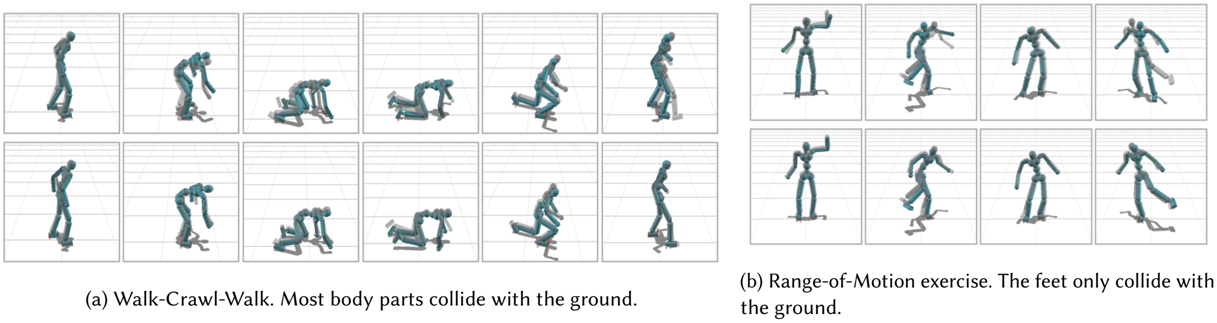
In this paper, we demonstrate that learning expert controllers for homogeneous sets of behaviors and combining them in a second training phase to create a combined controller allows us to have one, universal controller for a broad set of human motion. Below we


(a) Waltz

(b) Ballroom dancing

(c) Breakdancing

Fig. 9. Comparison of additive (top) and multiplicative (bottom) rewards.


(a) Walk-Crawl-Walk. Most body parts collide with the ground.

(b) Range-of-Motion exercise. The feet only collide with the ground.

Fig. 10. Comparisons of early termination methods. Early termination by collision with manual body assignment (top) and early termination by reward (bottom).

discuss limitations of this approach as well as promising avenues for future work.

We made a somewhat arbitrary decision to create eight clusters from the CMU motion capture database and therefore to create eight experts for inclusion in the MOE controller. It would be interesting to see whether larger numbers of clusters (and hence experts) would provide better performance. The expert performance should improve as the motion in the cluster becomes yet more homogeneous but the training of the MOE controller will be more difficult as the number of experts increases and at some point, it will fail to converge. Prior work in the literature indicates that more than 16 experts may present difficulties in learning MOE-like controllers [Peng et al. 2019; Starke et al. 2019].

Although our controller was very successful in reproducing most of the motions in the CMU motion capture database (and combinations of those motions created by a motion graph), that database does not of course span all of human motion. Further experiments would be necessary to see whether our approach of creating a monolithic

MOE controller can scale to yet larger datasets. There is presumably a limit on the number of experts that can be included without degrading overall performance or causing the learning of the combined controller to fail to converge. A further level in the hierarchy might be needed where a behavior selection algorithm chooses among a set of MOE controllers to select the one that best matches a given reference motion.

There are, of course, reference motions generated from the motion graph on which our controller fails. Further exploration would be needed to classify those failures but we expect that at least some of them may result from failures of the motion graph where the synthetic transitions created in the motion graph are not within the space of natural or physically realistic motion. Other failures are caused by the decrease in performance of the individual experts on their cluster of motions after the combined controller is trained. This represents the basic trade-off between specialization and the ability to have one combined controller that can be used on motions that contain behaviors from multiple clusters.

In the application of this work, it would be expected that the full range of desired behaviors might not be available at the outset. We did not explore network designs that would make it easier to incrementally add new motions and behaviors to the training set but that is a very interesting avenue for future work. We had hoped that beginners might fulfill this need, but our experiments all indicate that training beginners simultaneously with the existing MOE controller results in worse performance than having all motions represented by experts. It is possible that new behaviors and motions might be incorporated by first training new experts and then using the existing controller as a warm start for training a new controller that incorporates the new experts. If this approach for incrementally adding to the training set is successful, then controllers such as the one presented here might begin to play a role similar to that of *ImageNet* in the computer vision community. When a classifier is needed for a particular application, the standard practice now is to begin with a model trained on *ImageNet* and then refine the performance with additional training data for the particular application. The standard practice for controllers might be to build on a controller created from the CMU motion capture dataset or the full AMASS dataset and then refine it for the particular behaviors that were needed for a new application such as a video game focused on a particular sport.

REFERENCES

Amin Babadi, Kourosh Naderi, and Perttu Hämäläinen. 2019. Self-Imitation Learning of Locomotion Movements through Termination Curriculum. In *Motion, Interaction and Games, MIG 2019*. ACM, 21:1–21:7. https://doi.org/10.1145/3359566.3360072

Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The Option-Critic Architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 1726–1734. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14858

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

Glen Berseth, Cheng Xie, Paul Cernek, and Michiel van de Panne. 2018. Progressive Reinforcement Learning with Distillation for Multi-Skilled Motion Control. *CoRR* abs/1802.04765 (2018). http://arxiv.org/abs/1802.04765

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

Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *In Proc. of GDC*.

Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. 2018. Learning to Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 6, Article 179 (Dec. 2018). http://doi.acm.org/10.1145/3272127.3275048

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 (Dec. 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 (July 2010). http://doi.acm.org/10.1145/1778765.1781156

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

Danilo Borges da Silva, Rubens Fernandes Nunes, Creto Augusto Vidal, Joaquim B. Cavalcante Neto, Paul G. Kry, and Victor B. Zordan. 2017. Tunable Robustness: An Artificial Contact Strategy with Virtual Actuator Control for Balance. *Comput. Graph. Forum* 36, 8 (2017), 499–510. https://doi.org/10.1111/cgf.13096

Marco da Silva, Yeuhi Abe, and Jovan Popovic. 2008. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Comput. Graph. Forum* 27, 2 (2008), 371–380. https://doi.org/10.1111/j.1467-8659.2008.01134.x

Marco da Silva, Frédo Durand, and Jovan Popović. 2009. Linear Bellman Combination for Control of Character Animation. *ACM Trans. Graph.* 28, 3, Article 82 (July 2009). http://doi.acm.org/10.1145/1531326.1531388

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

Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. 2001. Composable Controllers for Physics-based Character Animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA. http://doi.acm.org/10.1145/383259.383287

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. 2017. Meta Learning Shared Hierarchies. *CoRR* abs/1710.09767 (2017). http://arxiv.org/abs/1710.09767

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

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

Matthew J. Hausknecht and Peter Stone. 2016. Deep Reinforcement Learning in Parameterized Action Space. In *4th International Conference on Learning Representations, ICLR 2016*. http://arxiv.org/abs/1511.04143

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

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3, 1 (1991), 79–87. https://doi.org/10.1162/neco.1991.3.1.79

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Trans. Graph.* 21, 3 (July 2002). http://doi.acm.org/10.1145/566654.566605

Taesoo Kwon and Jessica Hodgins. 2010. Control Systems for Human Running Using an Inverted Pendulum Model and a Reference Motion Capture Sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10)*. Eurographics Association, Goslar Germany, Germany. http://dl.acm.org/citation.cfm?id=1921427.1921447

Taesoo Kwon and Jessica K. Hodgins. 2017. Momentum-Mapped Inverted Pendulum Models for Controlling Dynamic Human Motions. *ACM Trans. Graph.* 36, 4, Article 145d (Jan. 2017). http://doi.acm.org/10.1145/3072959.2983616

Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. *ACM Trans. Graph.* 21, 3 (July 2002). http://doi.acm.org/10.1145/566654.566607

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

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

Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2009. Comprehensive Biomechanical Modeling and Simulation of the Upper Body. *ACM Trans. Graph.* 28, 4, Article 99 (Sept. 2009). http://doi.acm.org/10.1145/1559755.1559756

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 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 (Nov. 2014). http://doi.acm.org/10.1145/2661229.2661233

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 (June 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 (July 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 (May 2016). http://doi.acm.org/10.1145/2893476

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based Contact-rich Motion Control. *ACM Trans. Graph.* 29, 4, Article 128 (July 2010). http://doi.acm.org/10.1145/1778765.1778865

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graph.* 34, 6 (Oct. 2015). https://dl.acm.org/doi/10.1145/2816795.2818013

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

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*. OpenReview.net. https://openreview.net/forum?id=BJl6TjRcY7

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

Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: simulation and control of soft-bodied animals with biomimetic actuators. *ACM Trans. Graph.* 38, 6 (2019), 208:1–208:12. https://doi.org/10.1145/3355089.3356497

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust Physics-based Locomotion Using Low-dimensional Planning. *ACM Trans. Graph.* 29, 4, Article 71 (July 2010). http://doi.acm.org/10.1145/1778765.1778808

Igor Mordatch and Emo Todorov. 2014. Combining the benefits of function approximation and trajectory optimization. In *In Robotics: Science and Systems (RSS)*. http://www.roboticsproceedings.org/rss10/p52.html

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of Complex Behaviors Through Contact-invariant Optimization. *ACM Trans. Graph.* 31, 4, Article 43 (July 2012). http://doi.acm.org/10.1145/2185520.2185539

Masaki Nakada, Tao Zhou, Honglin Chen, Tomer Weiss, and Demetri Terzopoulos. 2018. Deep Learning of Biomimetic Sensorimotor Control for Biomechanical Human Animation. *ACM Trans. Graph.* 37, 4, Article 56 (July 2018). http://doi.acm.org/10.1145/3197517.3201305

Kensuke Onuma, Christos Faloutsos, and Jessica K. Hodgins. 2008. FMDistance: A Fast and Effective Distance Function for Motion Capture Data. In *Eurographics 2008 - Short Papers*. 83–86. https://doi.org/10.2312/egs.20081027

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

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 (July 2018). http://doi.acm.org/10.1145/3197517.3201311

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans. Graph.* 35, 4, Article 81 (July 2016). http://doi.acm.org/10.1145/2897824.2925881

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 (July 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. *CoRR* abs/1905.09808 (2019). http://arxiv.org/abs/1905.09808

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

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating Biped Behaviors from Human Motion Data. *ACM Trans. Graph.* 26, 3, Article 107 (July 2007). http://doi.acm.org/10.1145/1276377.1276511

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

Richard S. Sutton, Doina Precup, and Satinder P. Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* 112, 1-2 (1999), 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1

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

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

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 3540–3549. http://proceedings.mlr.press/v70/vezhnevets17a.html

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 (July 2012). http://doi.acm.org/10.1145/2185520.2185521

Ziyu Wang, Josh Merel, Scott Reed, Greg Wayne, Nando de Freitas, and Nicolas Heess. 2017. Robust Imitation of Diverse Behaviors. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., USA. http://dl.acm.org/citation.cfm?id=3295222.3295284

Jungdam Won and Jehee Lee. 2019. Learning Body Shape Variation in Physics-based Characters. *ACM Trans. Graph.* 38, 6, Article 207 (Nov. 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), 198:1–198:13. https://doi.org/10.1145/3130800.3130833

Jungdam Won, Jungnam Park, and Jehee Lee. 2018. Aerobatics control of flying creatures via self-regulated learning. *ACM Trans. Graph.* 37, 6 (2018), 181:1–181:10. https://doi.org/10.1145/3272127.3275023

Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst, and Michiel van de Panne. 2019. Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real. In *Proc. Conference on Robot Learning (CORL 2019)*. https://arxiv.org/abs/1903.09537

Yuting Ye and C. Karen Liu. 2010. Optimal Feedback Control for Character Animation Using an Abstract Model. *ACM Trans. Graph.* 29, 4, Article 74 (July 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 (July 2007). http://doi.acm.org/10.1145/1276377.1276509

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetric and Low-energy Locomotion. *ACM Trans. Graph.* 37, 4, Article 144 (July 2018). http://doi.acm.org/10.1145/3197517.3201397

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. *ACM Trans. Graph.* 37, 4 (2018), 145:1–145:11. https://doi.org/10.1145/3197517.3201366