Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning

LIBIN LIU
Disney Research
and
JESSICA HODGINS
Disney Research and Carnegie Mellon University

Given a robust control system, physical simulation offers the potential for interactive human characters that move in realistic and responsive ways. In this article, we describe how to learn a scheduling scheme that reorders short control fragments as necessary at runtime to create a control system that can respond to disturbances and allows steering and other user interactions. These schedulers provide robust control of a wide range of highly dynamic behaviors, including walking on a ball, balancing on a bongo board, skateboarding, running, push-recovery, and breakdancing. We show that moderate-sized Q-networks can model the schedulers for these control tasks effectively and that those schedulers can be efficiently learned by the deep Q-learning algorithm.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Control methods**; *Machine learning*; *Neural networks;*

Additional Key Words and Phrases: Human simulation, motion control, deep Q-learning

ACM Reference Format:

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 29 (June 2017), 14 pages.
DOI: http://dx.doi.org/10.1145/3083723

1. INTRODUCTION

Given a robust control system, simulation offers the potential for interactive human characters that respond naturally to the actions of the user or changes in the environment. The difficulty in capitalizing on this functionality has been in designing controllers for a variety of behaviors that are responsive to user input and robust to disturbances. Tracking a reference motion is an effective way to simplify the control design for complex human motions by leveraging the natural style and strategies contained in the motion data. However, tracking the motion sequence as it was recorded provides limited robustness and restricts the behaviors that are amenable to the approach.

A scheduler that reorders the tracking reference at runtime based on the state of the simulation can produce a robust control system. It not only synchronizes the reference with the simulation at specific events such as ground contact but also orchestrates the transitions among different control strategies, such as standing in place and taking steps, to respond to perturbations and user interactions. Handcrafted schedulers work well for locomotion [Lee et al. 2010a] and control systems using control graphs [Liu et al. 2016], but designing them often requires specific insights into the particular behaviors being controlled.

In this article, we develop a control system that takes advantage of automatically learned schedulers to achieve robust interactive control of a diverse set of behaviors. These schedulers arrange the control at the scale of short segments, each typically 0.1 seconds in length, called *control fragment*s. We model a scheduler with a medium-sized neural network, or Q-network, that maps a high-level representation of the state of the simulation to the best control fragment. We employ the deep Q-learning algorithm [Mnih et al. 2015] to train the scheduler by repeatedly executing an offline simulation. The learned scheduler performs in real time as a result of its compact formulation, enabling fast simulation of the behaviors.

The control fragments can be obtained by segmenting an input controller into small pieces for a behavior such as running or turning. We find that preserving as much as possible the original sequence in which the control fragments were arranged in the input controllers is important to the quality of the resulting motions, because it contains the control strategies that the human used to perform the motions. Our learned schedulers select out-of-sequence control fragments only when necessary.

This article makes three principal contributions: (1) We describe a scheduling scheme that can be learned to realize robust interactive control of a wide range of behaviors. The same mechanism can be used to manage transitions among a set of control strategies, allowing a successful response to larger perturbations and greater user control through a broader set of available actions. (2) We show that short control fragments allow schedulers to take actions immediately in response to new user commands or disturbances instead of waiting for predefined transition points. This quick response increases the robustness over that found in the original motion controllers. It also allows new transitions to be found that are not contained in the original controllers. (3) We adapt the deep Q-learning algorithm to allow efficient learning of the schedulers. A reward function that gives preference to the original sequence and an

Authors' addresses: L. Liu, 4720 Forbes Avenue, Lower Level, Suite 110, Pittsburgh, PA 15213; email: libin.liu@disneyresearch.com; J. Hodgins, the bulk of the research was done at Disney Research, and she is currently with Carnegie Mellon University, Robotics Institute, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: jkh@cs.cmu.edu.

DOI: http://dx.doi.org/10.1145/3083723

exploration strategy that gives more weight to the in-sequence control fragments significantly improves the efficiency of the learning and results in high-quality motions.

We demonstrate the power of this approach by controlling a diverse set of motions, including both motions that are amenable to a simple tracking approach such as running and turning as well as those that are not such as breakdance stunts, recovering from an unexpected push while standing, and motions on moving terrain such as a large ball, a bongo board, and a skateboard.

2. RELATED WORK

Designing controllers to realize complex human behaviors on simulated characters has a long history in computer animation. Numerous successful control systems have been proposed for a variety of behaviors, ranging from balancing [Macchietto et al. 2009; Hämäläinen et al. 2015] and locomotion [Yin et al. 2007; Coros et al. 2010; Mordatch et al. 2010], to dynamic aerial behaviors [Zordan et al. 2014; Ha and Liu 2014], breakdance [Al Borno et al. 2013] and bicycle stunts [Tan et al. 2014]. We refer interested readers to a survey article [Geijtenbeek and Pronost 2012] for an overview of this topic.

Tracking a reference sequence is a promising way to facilitate the control design and to simulate high-quality motions. A tracking controller is typically constructed from a reference motion clip and maintains a time-indexed target trajectory from which the control signals are computed. Open-loop tracking controllers can be built via trajectory optimization [Sok et al. 2007; Wampler and Popović 2009; Mordatch et al. 2012; Ha and Liu 2014] or sampling-based methods [Liu et al. 2010, 2015]. They usually lack robustness to unplanned disturbances because of the absence of feedback at runtime.

Feedback policies that compensate for disturbances at runtime can significantly improve the robustness of open-loop tracking controllers [Yin et al. 2007; Wang et al. 2010; Lee et al. 2010a; Muico et al. 2009], while longer-sighted control can be realized by simulating simplified models [Kwon and Hodgins 2010; Coros et al. 2010; Ye and Liu 2010; Mordatch et al. 2010; Kwon and Hodgins 2017]. Although most of these methods are designed for locomotion, they have also been applied to rotational and aerial behaviors [Zordan et al. 2014; Al Borno et al. 2014].

Recently, Liu and his colleagues [2016] demonstrated a generic method to learn robust tracking control for a variety of behaviors. Key to their success is a novel guided policy search method that efficiently learns time-varying linear feedback policies. Despite the success of this method in controlling motions on flat terrain, we find that it cannot achieve robust control in highly dynamic environments, such as the balance on a bongo board discussed in this article. These failures occur because its fixed control sequence cannot effectively handle drift due to disturbances from the environment. Instead, our tests demonstrate that the ability to reorder the control at runtime is essential to success with these tasks.

A fixed time-indexed reference-tracking scheme is brittle and often fails due to unplanned disturbances [Ye and Liu 2010; Lee et al. 2010a; Abe and Popovíc 2011]. A controller that uses a state-related index can deal with this issue. Switching the control strategies for the stance phase and the swing phase at every foot contact is a basic technique used by locomotion controllers [Nakanishi et al. 2004; Lee et al. 2010a]. For bipedal walking control, a phasing variable that is monotonic with respect to time is often employed to index the 2D or 3D gait control [Abe and Popovíc 2011; Buss et al. 2014]. For rotational behaviors, Zordan and his colleagues [2014] investigate several angular quantities that can effectively index the control policies. Our work also focuses on scheduling tracking control according to the simulation state. Instead of using a behavior-dependent index quantity, we build our scheduler on a high-level representation of the simulation state, which can be effectively used across behaviors.

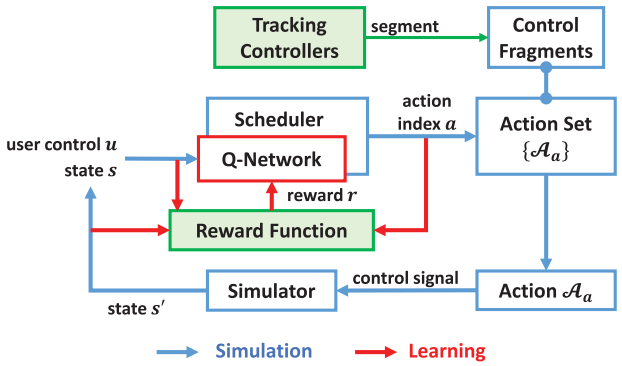


Fig. 1. System overview.

Reinforcement Learning (RL) provides a convenient framework for learning a control policy from past experience. In the context of character animation, value-based approaches have been successfully used in control systems with finite action sets [McCann and Pollard 2007; Treuille et al. 2007; Lee et al. 2010b; Coros et al. 2009]. Learning with a continuous action space is more difficult and often requires special treatment [Peng et al. 2015], where policy search methods play an important role in many approaches [Levine and Koltun 2013; Tan et al. 2014; Mordatch et al. 2015]. Recently, Mnih and his colleagues [2015] showed that a Deep Q-Network (DQN) can be effectively trained to perform at the same level as a human player across a number of classical video games. Their success stimulated research on applying DQN to various control problems [Lillicrap et al. 2015; van Hasselt et al. 2015; Nair et al. 2015; Peng et al. 2016]. Our scheduling problem has a continuous state space and a finite action set consisting of control fragments. This configuration allows us to utilize the deep Q-learning algorithm of Mnih et al. [2015] to learn the schedulers.

Learning can be performed on various building blocks. Interactive kinematic motion controllers have been learned on motion clips [Treuille et al. 2007], motion fragments [McCann and Pollard 2007], or motion fields [Lee et al. 2010b]. In the context of robotics and physics-based character animation, learning can be done at the scale of every timestep [Liu et al. 2013], while motion primitives that accomplish a complete task such as grasping an object or walking one step [Stulp et al. 2012; Coros et al. 2009; Peng et al. 2015] are widely used as the building block. In this article, we evenly segment a tracking controller into short control fragments using a fixed time interval independent of the behavior being controlled. These control fragments can be viewed as motion primitives. Since they are of uniform length rather than being segmented at contact or behavior changes, the domain knowledge in the motion primitive is reduced.

3. SYSTEM OVERVIEW

Our goal is to realize robust control of a diverse set of behaviors while allowing the user to interact with the simulated character. To achieve this goal, our control system learns an individual scheduler for each task that periodically reschedules the control fragments according to the state of the simulation. Figure 1 provides

an overview of the major components of our system, where the simulation pipeline is shown with blue arrows, and the additional procedures for the offline learning process are drawn in red.

The input to our system is control fragments for the target behaviors and a reward function that provides a high-level description of the task. These control fragments are created in a preprocessing stage by segmenting a precomputed tracking controller into a series of short pieces, each 0.1s in length. The input tracking controllers are typically constructed from a motion clip and each controls a complete movement such as a running cycle. These control fragments collectively compose an action set $A = \{\mathcal{A}_a\}$, where every action a corresponds to a control fragment $\mathcal{A}_a$. A scheduler maintains a medium-sized artificial neural network, or a Q-network, that computes the long-term reward of taking an action at a state. The simulation pipeline starts from the evaluation of the Q-network according to the current simulation state and the task parameters. The control fragment having the highest long-term reward is selected and computes the control signals to actuate the character in the following 0.1s. When this control fragment finishes, another control fragment is scheduled according to the new state.

Our system utilizes the deep Q-learning algorithm [Mnih et al. 2015] to train the Q-networks offline. Starting from a randomly initialized Q-network, the learning process repeatedly executes the simulation pipeline and collects simulation experiences to evolve the Q-network toward a better approximation. Unlike an online simulation where the scheduler always takes the best action, an exploration strategy is used in the learning process that selects nonoptimal control fragments probabilistically to explore the state-action space. Once the selected control fragment finishes, the simulation is evaluated by the reward function, and the Q-network is updated accordingly.

4. LEARNING OF SCHEDULERS

We formulate our scheduling problem as a Markov Decision Process (MDP) represented by a tuple (X, A, T, R, γ), which consists of a state space, X, an action set, A, a transition function, T, a reward function, R, and a discount factor, γ. Our problem has a hybrid state space, where a state $x = (s, u, \tilde{a}) \in X$ models the continuous simulation state $s \in S$, the optional task parameter $u \in U$, and the index of the previous action $\tilde{a} \in A$. Taking an action $a \in A$ at state x leads to a state transition, $T : (x, a) \mapsto x'$,

$$T : \left(\begin{bmatrix} s \\ u \\ \tilde{a} \end{bmatrix}, a \right) \mapsto \begin{bmatrix} s' \\ u' \\ \tilde{a}' \end{bmatrix} = \begin{bmatrix} \mathcal{A}_a(s) \\ u \\ a \end{bmatrix}, \tag{1}$$

where $s' = \mathcal{A}_a(s)$ represents the simulation under the control of the control fragment $\mathcal{A}_a$. The reward function $r = R(x, a, x')$ evaluates this state transition and determines how it fulfills a given task. A control policy, or in our case, a scheduler, $\pi : X \to A$, defines a mapping from the state space to the action set. Starting from a state x, repeatedly executing a control policy π leads to a transition sequence $\{x_0 = x, a_0, r_0, x_1, a_1, r_1, \dots\}$ that satisfies $a_t = \pi(x_t)$, $x_{t+1} = T(x_t, a_t)$, and $r_t = R(x_t, a_t, x_{t+1})$. Then the evaluation of π at state x is given by a discounted accumulative reward $V^\pi(x) = \sum_{t=0}^{\infty} \gamma^t r_t$ over the transition sequence. The discount factor γ implicitly determines the planning horizon. We use $\gamma = 0.95$ for all of the results presented here.

Solving a MDP problem means figuring out the optimal control policy having the maximal accumulative reward at all possible states. Q-learning [Watkins 1989] is a class of model-free methods that solve an MDP problem by evolving a Q-value function defined as $Q^\pi(x, a) = r + \gamma V^\pi(x')$. The optimal Q-value function recursively satisfies the Bellman equation:

$$Q^*(x, a) = r + \gamma \max_{a'} Q^*(x', a'). \tag{2}$$

Once it is found, the optimal policy π^* can be simply derived as

$$\pi^* : x \mapsto \arg\max_a Q^*(x, a). \tag{3}$$

The hybrid state space in our problem necessitates the use of a parameterized Q-value function $Q(x, a; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the parameters. Our system employs an artificial neural network to approximate this function and train it using the deep Q-learning algorithm [Mnih et al. 2015]. For simplicity, we call this neural network a Q-network hereafter.

4.1 States

We model our simulated character as an underactuated articulated rigid body system with an unactuated root joint mounted on the character's pelvis. We use a combination of two sets of selected properties, $s = s_m \cup s_e$, to represent the state of the simulation, where s_m models the character's movement and s_e captures the state of the objects in the environment that interact with the character. All of these properties are measured in reference coordinates that move horizontally with the character and have an axis aligned with the character's facing direction. While there are many candidate properties, we follow the choice of Liu et al. [2016] that sets $s_m = (\boldsymbol{q}_0, h_0, \boldsymbol{c}, \dot{\boldsymbol{c}}, \boldsymbol{d}_l, \boldsymbol{d}_r, \boldsymbol{L})$. This 18-dimensional vector contains the orientation $\boldsymbol{q}_0$ and height h_0 of the root joint, the centroid position $\boldsymbol{c}$ and velocity $\dot{\boldsymbol{c}}$, the vectors pointing from the Center of Mass (CoM) to the centers of left foot $\boldsymbol{d}_l$ and right foot $\boldsymbol{d}_r$, and the angular momentum $\boldsymbol{L}$. The environmental state s_e is defined for each behavior. For example, to achieve a stable oscillation on a bongo board, our control system observes the relative position of the bongo board's wheel with respect to the character's CoM and the velocity of the wheel. An empty environmental state $s_e = \emptyset$ is used for the behaviors that do not interact with a moving object, such as running and breakdancing. We defer detailed definitions of s_e for each behavior until Section 5.

The task parameter $u \in U$ consists of the parameters that a user can interactively control at runtime. For example, when controlling the direction in which the character runs or skateboards, we choose the angle between the current direction and the target direction as the task parameter. If a task does not have a controllable parameter, such as balancing on a bongo board, we choose $U = \emptyset$. We will discuss the separate definitions of these task parameters in Section 5.

A state vector $x = (s, u, \tilde{a})$ in our MDP problem includes both the simulation state, s, and the task parameter, u. It additionally records the index of the previous action, $\tilde{a}$, for identifying an in-sequence action as discussed in the next section. The scheduler only takes s and u into consideration when selecting the next action, where each dimension of the simulation state s is centralized and scaled according to the mean and standard variance of the reference motion clips from which the input tracking controllers are constructed, and the task parameter u is normalized according to the range of its possible values.

4.2 Actions

The action set A consists of a number of control fragments, which can be extracted from precomputed tracking controllers. A tracking controller typically maintains a time-indexed reference trajectory and computes control signals from it. By cutting the reference trajectory into small pieces, our control system creates a series of short fragments of the original tracking control, which we call *control*

fragments. If the original tracking controller has associated feedback policies, we also embed them into the corresponding control fragments. Every scheduler in our system takes a collection of these control fragments obtained from one or more input controllers as the action set. Each time a scheduler selects a control fragment, the control system is set to the reference time corresponding to that control fragment.

Our control system segments a tracking controller with an interval, δt, predefined according to the length of its reference trajectory. Although the exact value of δt is not critical, if it is too long, the scheduler will become unresponsive to changes. If it is too short, the large number of actions will increase the complexity in learning the scheduler. Inspired by Liu et al. [2016], we choose $\delta t = 0.1s$.

Executing a control fragment means applying its associated feedback policy, computing the control signals, and advancing the simulation by δt seconds. We represent this process as

$$s' = \mathcal{A}(s), \tag{4}$$

where $\mathcal{A}$ represents the control fragment being executed, while s and s' are the simulation states before and after the execution, respectively. At runtime, the scheduler selects a new control fragment according to s' after $\mathcal{A}$ finishes.

The segmentation of the input tracking controllers suggests a reference sequence $O = \{\langle \tilde{a}, a\rangle\}$, where an *action pair* $\langle \tilde{a}, a\rangle$ indicates that control fragment $\mathcal{A}_{\tilde{a}}$ is followed by $\mathcal{A}_a$ in an input tracking controller. Hereafter, we refer to an action a as an *in-sequence* action of action $\tilde{a}$ if $\langle \tilde{a}, a\rangle \in O$; and otherwise, it is an *out-of-sequence* action.

4.3 Reward Function

A reward function $R : X \times A \times X \to \mathbb{R}$ specifies the task that the scheduler is designed to accomplish. In this article, the reward function is a summation of four penalty terms:

$$R(x, a, x') = E_{\text{tracking}} + E_{\text{preference}} + E_{\text{feedback}} + E_{\text{task}} + R_0, \tag{5}$$

where R_0 is a default reward returned when all of the requirements of the task are satisfied; otherwise the penalties are applied, and the final reward is less than R_0. We use $R_0 = 5$ for all our results.

Tracking: Our system trains a scheduler to follow the reference sequence O in order to produce high-quality motions. The tracking penalty term E_{tracking} of the reward function thus penalizes any out-of-sequence action by

$$E_{\text{tracking}}(x, a, x') = \begin{cases} 0 & \langle \tilde{a}, a\rangle \in O \text{ or } \tilde{a} \notin \tilde{O} \\ -d_o & \text{otherwise,} \end{cases} \tag{6}$$

where $d_o > 0$ is a constant penalty, $\tilde{O} = \{\tilde{a} : \exists a, \langle \tilde{a}, a\rangle \in O\}$. This term gives strong preference to the reference sequence while still allowing out-of-sequence actions when necessary. We use $d_o = 2.0$ for all our results.

Action Preference: The term $E_{\text{preference}}$ reflects the user's preference when a task can be accomplished by multiple actions. Let A_I represent the subset of favored actions in A; we define

$$E_{\text{preference}} = \begin{cases} 0 & a \in A_I \\ -d_p & \text{otherwise,} \end{cases} \tag{7}$$

where $d_p > 0$ is a constant penalty. We use $d_p = 0.2$ unless specified otherwise.

Feedback: The feedback term E_{feedback} penalizes excessive feedback when the control fragments have associated feedback policies. Excessive feedback happens either because a failure has occurred that the feedback policy cannot handle or because an improper control fragment has been selected. In either case the controller produces poor-quality results.

Task: The task term E_{task} models task-dependent penalties, such as a penalty applied when the character fails to move in a target direction. We will describe individual tasks and the corresponding task terms in Section 5.

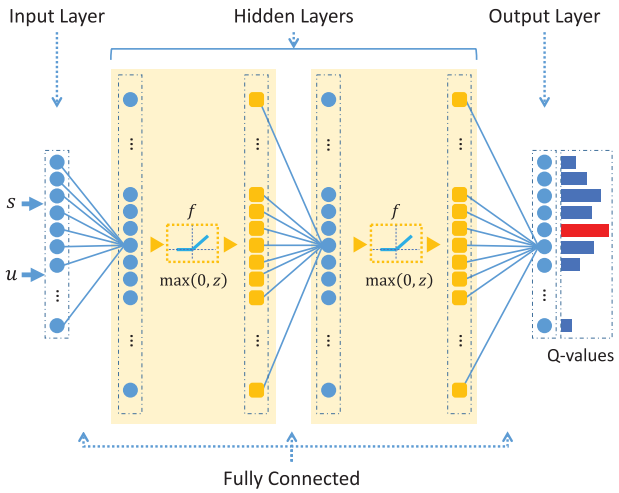


Fig. 2. Q-Network: The input layer is a state vector that models the simulation state s and the task parameter u; the output layer computes the Q-values of all actions taken at the input state; two hidden layers consist of Rectified Linear Units (ReLUs) whose activation function is $f = \max(0, z)$, where z is the input to the unit. All the layers are fully connected.

4.4 Q-Network

We use a feedforward artificial neural network to approximate the Q-value function, $Q(x, a; \boldsymbol{\theta})$, defined in Equation (2). Instead of computing the Q-value for one state-action pair (x, a) at a time, the limited number of actions allows us to compute Q-values for all of them simultaneously with a compact network structure, as in Figure 2. The input layer of this Q-network is a vector that consists of the simulation state, s, and the task parameter, u. The output layer is a vector of dimension $|A|$, that is, the number of control fragments, whose a-th component corresponds to the Q-value of action a taken at state x. With this representation, the true Q-value function can be written with index notation as

$$Q(x, a; \boldsymbol{\theta}) = [\tilde{Q}(x; \boldsymbol{\theta})]_a. \tag{8}$$

This structure is inspired by the work of Mnih and his colleagues [2015]. Unlike the deep networks that they found were required for learning video game strategies from images, we find that relatively shallow networks can successfully model the schedulers for the tasks tested in this article. Each of our Q-networks has two fully connected hidden layers, and every hidden layer consists of 300 ReLUs whose activation function, f, is defined as

$$f(z) = \max(0, z), \tag{9}$$

where z is the scalar input to a unit. The network parameter $\boldsymbol{\theta}$ thus contains the weights and bias terms of both hidden layers and the output layer.

ALGORITHM 1: Learning of Q-network

```
initialize D ← ∅
initialize a Q-network with random parameters θ
backup current parameters θ̂ = θ
loop
    choose a starting state x_0 = (s_0, u_0, ã_0)
    t ← 0
    while x_t ∉ X_fail and t < T_episode do
        select an action a_t according to x_t = (s_t, u_t, ã_t):
            with probability ε_r select a random action
            with probability ε_o select a_t s.t. ⟨ã_t, a_t⟩ ∈ O
            otherwise select a_t = argmax_a Q(x_t, a, θ)
        x_{t+1} ← T(x_t, a_t); r_t ← R(x_t, a_t, x_{t+1})
        store transition tuple (x_t, a_t, r_t, x_{t+1}) in D
        update θ with batch stochastic gradient descent
        every N_backup steps backup θ to θ̂
        t ← t + 1
    end while
end loop
```

4.5 Deep Q-Learning

Our control system learns separate schedulers for each task. Each scheduler consists of a Q-network with different numbers of input and output units. The learning algorithm iteratively updates the network parameters $\boldsymbol{\theta}$ via small steps that minimize a loss function

$$L(\boldsymbol{\theta}) = \mathrm{E}_{x,a,x'}[||y(x,a,x';\hat{\boldsymbol{\theta}}) - Q(x,a;\boldsymbol{\theta})||^2] + w_r||\boldsymbol{\theta}||^2, \quad (10)$$

where the regularization term is weighted by $w_r = 0.001$. Unlike an ordinary regression problem, the target function

$$y(x,a,x';\hat{\boldsymbol{\theta}}) = \begin{cases} r + \gamma \max_{a'} Q(x',a';\hat{\boldsymbol{\theta}}) & x' \notin X_{\text{fail}} \\ 0 & x' \in X_{\text{fail}} \end{cases} \quad (11)$$

changes when the current parameters $\hat{\boldsymbol{\theta}}$ are updated. The terminal set X_{fail} contains the states in which the control will inevitably fail, for example, the character is falling.

The learning algorithm updates the network parameters $\boldsymbol{\theta}$ after every transition step using a batch stochastic gradient descent method. The loss function of Equation (10) is evaluated over a minibatch consisting of $N_{\text{batch}} = 50$ transition tuples randomly selected from a sample set $D = \{(x_i, a_i, r_i, x_i')\}$, which stores up to $N_D = 10^6$ most recent transition tuples. The update rule can be written as

$$\theta = \theta - \alpha \frac{L'_\theta}{L_0}, \quad (12)$$

where α is the learning rate and L'_θ is the derivative of the loss function with respect to θ, which can be efficiently computed through backpropagation. A variation of the RMSprop algorithm [Tieleman and Hinton 2012] is used to scale the gradients as suggested by Mnih et al. [2015]. The scale factor is computed as

$$L_0 = \sqrt{\mathrm{MA}[{L'_\theta}^2] - (\mathrm{MA}[L'_\theta])^2 + \delta_0}, \quad (13)$$

where $\mathrm{MA}[z] = (1.0 - \beta)\,\mathrm{MA}[z] + \beta z$ is the moving average of a quantity z with decay factor β, and δ_0 is a small constant for avoiding division by zero. We use $\beta = 0.05$ and $\delta_0 = 0.01$ for Equation (13). We use a learning rate $\alpha = 2.5 \times 10^{-5}$ at the beginning of the learning process and halve it every two million steps.

Instead of updating the target function of Equation (11) with $\boldsymbol{\theta}$ in every learning step, Mnih and his colleagues [2015] suggest that keeping $\hat{\boldsymbol{\theta}}$ unchanged for a fixed N_{backup} steps reduces the variation of the target function and improves the stability of the learning algorithm. We adopt this idea and use $N_{\text{backup}} = 5{,}000$ in all our experiments.

Algorithm 1 outlines the major steps of the learning process. Starting from a randomly initialized Q-network, the outer loop of the algorithm repeatedly generates episodes of simulation and updates the parameters until a successful scheduler is found. Each simulation episode begins with a chosen starting state x_0. The inner loop of the algorithm iteratively elongates the episode by selecting an action a_t according to the current state x_t, executing the corresponding control fragment $\mathcal{A}_{a_t}$ to advance the state to x_{t+1}, computing the immediate reward r_t and storing the transition tuple (x_t, a_t, r_t, x_{t+1}) in the sample set D, and updating the parameters $\boldsymbol{\theta}$ with the batch stochastic gradient decent algorithm described previously. The simulation episode ends when either the state x_t is in the terminal region X_{fail} or the maximal length of $T_{\text{episode}} = 150$ transition steps is reached.

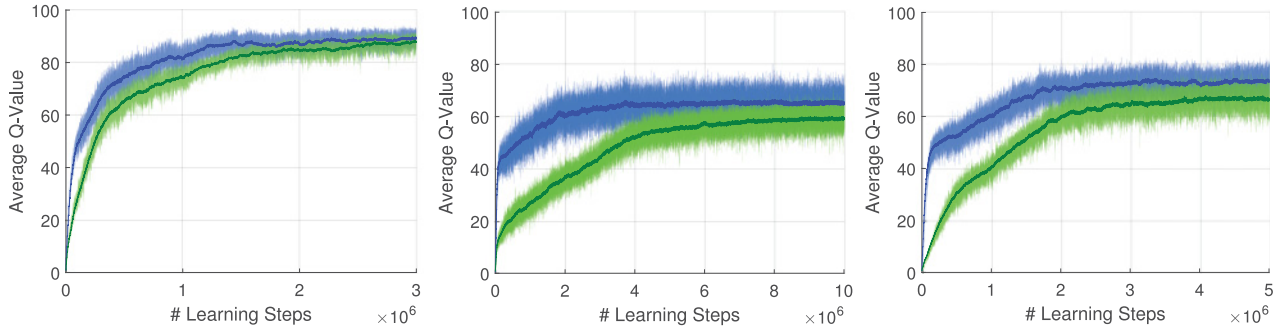
We create the starting state $x_0 = (s_0, u_0, \tilde{a}_0)$ for the first episode by randomly picking a simulation state s_0 from those collected during the construction of the input tracking controllers, setting the associated action $\tilde{a}_0$ to be consistent with s_0, and, if applicable, assigning a random task parameter u_0. If an episode ends in the terminal region, we roll back 20 transition steps and start a new episode from that simulation state with a new task parameter u_0. If the episode fails too soon or ends without failing, the new starting state is chosen in the same way as the first episode.

The action a_t is chosen in a ε-greedy fashion: with probability ε_r, the random exploration strategy is applied and a random action is selected; with probability ε_o, the tracking exploration strategy is applied and an in-sequence action a_t that satisfies $\langle \tilde{a}_t, a_t \rangle \in O$ is selected; otherwise, the action $a_t = \mathrm{argmax}_a\, Q(x_t, a, \boldsymbol{\theta})$ is selected, which exploits the current scheduler. In all of our experiments, the probability ε_r is fixed to 0.1, while ε_o is linearly annealed from 0.9 to 0.1 in the first $N_{\text{A}} = |A| \times 10\text{k}$ steps and is fixed at 0.1 thereafter. We find that the tracking exploration strategy significantly accelerates the learning process, as indicated in Figure 3. The blue curves in Figure 3 correspond to the learning processes using our exploration strategy, while the green curves show the learning processes using the same configurations except that the tracking exploration is disabled by setting $\varepsilon_o = 0$. With the latter settings, the learning processes can easily get stuck in local optima, and the learned schedulers often select unnecessary out-of-sequence actions, resulting in jerky movements.

When performing a control fragment $\mathcal{A}_a$, the learning algorithm applies noise torques $\tau_\epsilon \sim \mathcal{N}(0, \sigma_\tau^2)$ to every Degree of Freedom (DoF) of the character's legs and waist, where the noise level $\sigma_\tau = 5\text{Nm}$. This procedure forces the learning algorithm to visit different states under the same action sequence and allows the learned scheduler to deal with larger uncertainty as suggested by Wang et al. [2010] and Liu et al. [2016]. The computation cost prohibits a complete cross validation on the hyperparameters described previously. Instead, we choose these parameters empirically on the task of balancing on a bongo board and use the same values for all other tasks.

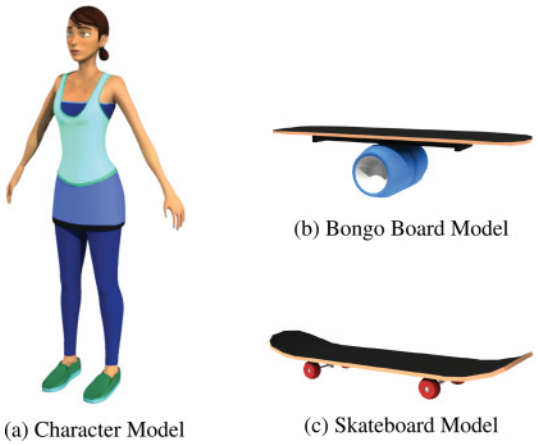
5. RESULTS

We tested our system using the character model shown in Figure 4(a), which is 1.7m tall and weighs 61kg. It has 51 DoF in total, including a 6-DoF unactuated root. Most of our results are tested using the control fragments that compute joint-level control signals using PD-servos. We use PD-gains $k_p = 500$, $k_d = 50$ for all of the joints except for the nearly passive toes and wrists, for which we use $k_p = 10$, $k_d = 1$.



(a) Balancing on a Bongo Board (b) Skateboarding (c) Running

Fig. 3. Learning curves of (a) the bongo board balancing task, (b) the skateboarding task, and (c) the running task, using feedback-augmented control fragments. The filled areas represent the average Q-values $V^\theta = \mathrm{E}_{x,a}\left[Q(x,a;\boldsymbol{\theta})\right]$ computed upon the minibatches used in every learning step. The solid curves represent the moving average $\mathrm{MA}[V^\theta] = (1-\beta)\mathrm{MA}[V^\theta] + \beta V^\theta$ with decay factor $\beta = 0.01$. The learning processes shown in blue use our exploration strategy, while those drawn in green have the tracking exploration disabled.



(a) Character Model (b) Bongo Board Model (c) Skateboard Model

Fig. 4. Models used in this article.

We have implemented our system in C++. We augment the Open Dynamic Engine (ODE) with an implicit damping scheme [Tan et al. 2011] as described in Liu et al. [2013] to simulate the character, enabling the use of a simulation timestep of 0.01s. This large timestep significantly speeds up the learning process and allows real-time performance. The learned Q-networks are executed by an implementation based on the Eigen library [Guennebaud et al. 2010]. The simulation pipeline runs at 30× real time on a desktop with an Intel Core i7-3770 @ 3.4GHz CPU.

The training routine is implemented in Python 2.7 based on the Theano library [Bastien et al. 2012; Bergstra et al. 2010], which provides an off-the-shelf mechanism to compute gradients and update parameters. The training is performed on the same CPU and usually requires several hours of computation. We check the learned scheduler every one million iterations and stop the learning process if a successful scheduler is found. Table I provides detailed performance statistics for all of the tested tasks.

5.1 Scheduling of Open-Loop Fragments

We start our experiments by learning schedulers for open-loop control fragments. Because of the absence of feedback, this type of tracking controller can only produce a single motion from a designated starting state. We will show that our system can achieve robust control of several difficult behaviors by scheduling those open-loop control fragments.

We prepare the input open-loop tracking controllers with the SAMCON algorithm [Liu et al. 2010, 2016], which is a sampling-based method that constructs controllers from reference motion capture clips. The output of SAMCON is a target trajectory that can be tracked with PD-servos to reproduce the input motion clip. After the segmentation process, each control fragment contains a short piece of the target trajectory, producing a short clip of the target behavior. Because these control fragments have no associated feedback policies, we use $E_{\text{feedback}} = 0$ for all of them.

5.1.1 *Balancing on a Bongo Board.* In this task, the character tries to maintain balance on the bongo board shown in Figure 4(b), which consists of a board 80cm in length, a wheel 12cm in diameter, and a track mounted under the board constraining the wheel to move along it. The total weight of this bongo board is 3.2kg.

Stable contacts are very important to the success of this task; however, we find that our character's rigid feet do not remain on the board. The same problem has been investigated by Jain and Liu [2011], who suggested the use of soft deformable feet to mitigate the problem, at the cost of increased simulation time. Instead, we attach the inner side of both feet to the board with a pin joint.

The environmental state $s_e = \{\boldsymbol{v}_{\text{wheel}}, \boldsymbol{d}_{\text{wheel}}\}$ for the bongo board balancing task models the velocity of the wheel, $\boldsymbol{v}_{\text{wheel}}$, and the relative position between the wheel and the character's CoM, $\boldsymbol{d}_{\text{wheel}}$. We do not include user control in this task and set $U = \emptyset$. The task term of the reward function penalizes the horizontal deviation between the character's CoM and the wheel using

$$E_{\text{task}} = -f(||\boldsymbol{d}^*_{\text{wheel}}|| - 0.1), \tag{14}$$

where the function f is the rectifier defined in Equation (9), and $\boldsymbol{d}^*_{\text{wheel}}$ represents the horizontal components of $\boldsymbol{d}_{\text{wheel}}$.

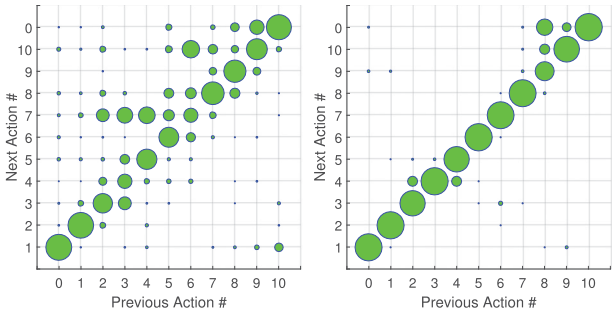
Our system constructs an open-loop tracking controller from a reference motion capture clip where the actor oscillates on the board. After segmentation, the action set contains 11 open-loop control fragments that collectively reproduce one cycle of the reference oscillation. The learned scheduler allows the character to maintain balance on the board without external perturbations.

Figure 5(a) provides an analysis of how the learned scheduler works, where each data point (i, j) corresponds to an action pair $\langle a_i, a_j \rangle$ indicating that the action a_j on the vertical axis is taken after

Table I. Performance Statistics

Task	dim[X] dim[S]	dim[X] U	$\|A\|$	# Steps ($\times 10^6$)	t_{learning} (hour)	MA[V^{θ}]
bongo board balancing (*)	24	$\emptyset$	11	2	4.0	67.0
walking on a ball (*)	24	$\{\phi\}$	22	3	4.7	41.4
walking on a ball using raw states of rigid bodies (*)	126	$\{\phi\}$	22	3	5.2	47.5
bongo board balancing	24	$\emptyset$	11	2	3.9	87.3
bongo board balancing using raw states of rigid bodies	126	$\emptyset$	11	2	4.3	90.8
bongo board balancing w/push	24	$\emptyset$	51	4	7.7	83.5
skateboarding	24	$\{\phi\}$	39	4	8.3	63.8
skateboarding – tic-tac	24	$\{\phi\}$	39	4	8.2	54.7
running	18	$\{\phi\}$	50	3	4.2	73.0
push-recovery	18	$\emptyset$	117	4	7.1	88.3
breakdancing	18	$\{A_0, A_1\}$	146	5	8.8	54.3
breakdancing w/sitting up	18	$\{A_0, A_1\}$	36	3	4.7	77.5
push-recovery w/balancing control	18	$\emptyset$	109	4	8.3	84.8

Tasks marked with (*) are learned on open-loop control fragments. dim[S] represents the total DoF of both the movement state s_m and environmental state s_e. U represents the user control parameters. dim[U] = 0 if U is empty or otherwise 1. $|A|$ represents the number of actions. # steps and t_{learning} are the total number of learning steps and the learning time, respectively. MA[V^{θ}] is the average Q-value when the learning process ends.



(a) Open-loop Fragments (b) Feedback-augmented Fragments

Fig. 5. Frequency of taking an action after another action for a bongo board scheduler. The radius of a data point is proportional to the frequency with which the action on the Y axis follows the action on the X axis. The diagonal represents the in-sequence action pairs. Left: A scheduler learned on open-loop control fragments. Right: A scheduler learned on feedback-augmented control fragments.

the action a_i on the horizontal axis. The radii of these data points are proportional to the frequencies with which the action pairs are taken. With the actions indexed such that the diagonal of the graph represents the in-sequence action pairs, Figure 5(a) indicates that the learned scheduler frequently takes out-of-sequence actions in this task, thus violating the cycle embedded in the reference order. Although the character can stay on the board without falling, it cannot reproduce the reference oscillation.

5.1.2 *Walking on a Ball.* In this task, we let the character walk on a ball and move in a user-specified direction. The ball is 0.7m in diameter and weighs 18kg. We use $s_e = \{\boldsymbol{v}_{\text{ball}}, \boldsymbol{d}_{\text{ball}}\}$ as the environmental state, which captures the velocity of the ball, $\boldsymbol{v}_{\text{ball}}$, and the vector pointing from the character's CoM to the center of the ball, $\boldsymbol{d}_{\text{ball}}$. The task parameter $U = \{\phi\}$ is the angle between the current direction of motion and the target direction. We define the task term E_{task} of the reward function as

$$E_{\text{task}} = E_{\text{CoM}} + E_{\text{direction}}, \tag{15}$$

where the balance term $E_{\text{CoM}} = -||\boldsymbol{d}^*_{\text{ball}}||$ keeps the character's CoM above the ball with $\boldsymbol{d}^*_{\text{ball}}$ representing the horizontal components of $\boldsymbol{d}_{\text{ball}}$, and the direction term $E_{\text{direction}}$ controls the direction of motion by

$$E_{\text{direction}} = -f(\epsilon_c - ||\dot{\boldsymbol{c}}^*||) - f(\delta\phi - \epsilon_\phi). \tag{16}$$

The first penalty term of Equation (16) takes effect when the character moves slower than $\epsilon_c = 0.1$m/s, where $\dot{\boldsymbol{c}}^*$ represents the horizontal components of the character's centroid velocity $\dot{\boldsymbol{c}}$. The second term of Equation (16) penalizes the directional error if it exceeds a threshold $\epsilon_\phi = 5^\circ$.

The input open-loop tracking controller is constructed from a short motion capture clip in which the actor walks a few steps on a ball and moves forward by rolling the ball underneath its feet. The action set contains 22 open-loop control fragments. Although there are no reference clips for turns, the learned scheduler allows the character to stably walk on the ball and slowly turn to a user-controlled direction.

To further evaluate the generality of the learning algorithm, we learn a new scheduler to allow the character to walk on a big ball of 2.0m in diameter and 42kg as shown in Figure 6(g). The open-loop control fragments learned on the 0.7m ball are reused in this task. Without any other modification, the learning process automatically finds a successful scheduler using these control fragments, even though the environment has dramatically changed.

5.2 Scheduling of Feedback-Augmented Fragments

As shown in the last section, when the scheduler is learned using open-loop fragments, the lack of feedback necessitates the frequent use of out-of-sequence actions. This issue often leads to poor-quality motions on cyclic behaviors such as running, because the scheduler has to repeatedly break the motion cycles. Liu et al. [2016] suggests that every open-loop control fragment can be enhanced with an associated linear feedback policy. Such a feedback-augmented control fragment can stabilize the simulation within the vicinity of the reference motion, so long as the starting state falls into the basin of attraction of the associated feedback policy. We will show that a high-quality scheduler can be learned using these feedback-augmented control fragments.

When a control fragment $\mathcal{A}_a$ is performed, its associated feedback policy computes a corrective offset, Δ_a, according to the current simulation state. Δ_a contains the additional rotations on several selected joints, including the hips, the knees, and the waist. It will be applied to every frame of the target trajectory of the control

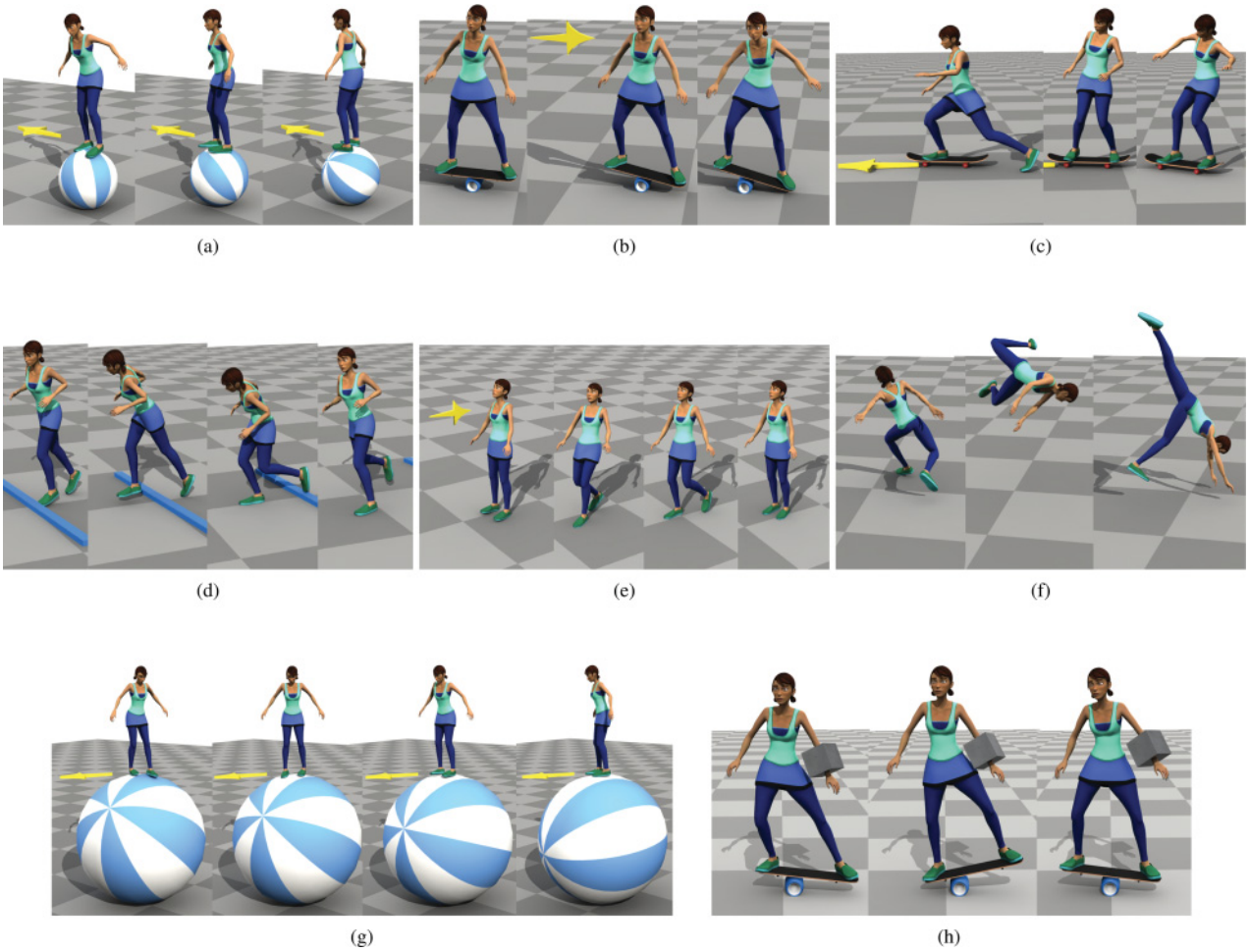


Fig. 6. Real-time simulation of the learned schedulers. (a) A character walks on a ball. (b) A character balances on a bongo board and touches the board with the ground to regain balance after a push. (c) A skateboarder pushes off the ground, rolls on the board, and turns to the right. (d) A runner trips over a bump and recovers. (e) A character steps back to regain balance after a push. (f) A breakdancer performs a backflip. (g) A character walks on a big ball. (h) A character balances on a bongo board with a heavy object on her left arm.

fragment. The feedback term E_{feedback} is thus defined as

$$E_{\text{feedback}} = -f(||\Delta_a|| - \epsilon_{\text{F}}) \tag{17}$$

for these control fragments, where ϵ_{F} is a constant threshold that indicates a normal range of feedback. We use $\epsilon_{\text{F}} = 0.2$ radians for all the results in the following.

To further facilitate the learning process, we let the random exploration strategy only choose a control fragment whose feedback is less than a threshold $\Delta_{\max}$. During the tracking exploration procedure and the exploitation procedure, if the selected control fragment asks for a corrective offset exceeding $\Delta_{\max}$, we assume that the control has failed and consider the current state as a terminal state. We use $\Delta_{\max} = 1.5$ radians unless specified otherwise.

5.2.1 *Balancing on a Bongo Board with Feedback-Augmented Fragments.* We let our control system learn a new scheduler using the feedback-augmented control fragments derived from the open-loop fragments used in the last section. The feedback policies associated with those control fragments cannot accomplish the balancing task without rescheduling, and the character falls off of the board in 5s. The failure occurs because the single-wheel support is unstable in nature and the simulation can easily drift out of the basin of attraction of the next control fragment if the schedule is determined solely from the reference timing. In contrast, the learned scheduler can achieve a stable oscillation by automatically selecting the appropriate control fragment according to the current state.

We performed the same experiments as discussed in the last section on the new scheduler. The result is shown in Figure 5(b). Unlike the scheduler learned on the open-loop control fragments (Figure 5(a)), Figure 5(b) indicates that the reference timing drives the new scheduler most of the time, but a small number of out-of-sequence action pairs appear occasionally and are important to the success of the control. Most of these out-of-sequence pairs occur around Action 3 and Action 8, both of which are taken near the highest points of the oscillation. This result indicates that the freedom to choose appropriate out-of-sequence actions is critical to the success of this controller, while the feedback policies help the action pairs stay in the reference sequence, improving the motion quality. We encourage readers to watch the supplemental video for better comparison.

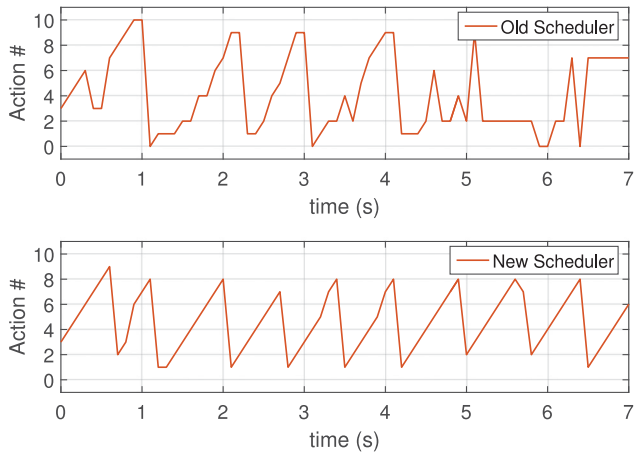


Fig. 7. Action sequences for the bongo board task with a heavy object on the character's left arm. Top: The scheduler learned on the normal character is used. Bottom: The scheduler is learned for this new task.

The scheduler learned earlier can maintain a stable oscillation on the board, but moderate sideways pushes will cause the character to fall over. To handle these perturbations, we additionally include an auxiliary motion where the character regains balance by touching the board on the ground and then resumes the balancing task. We find that the method of Liu et al. [2016] cannot be directly applied here because there is not a predefined transition between the auxiliary motion and the oscillation. To solve this problem, we enhance the original method and allow out-of-sequence control fragments to be taken with the probability 0.1 during the construction process. The final action set includes both the new control fragments and their mirror actions, which makes the total number of actions 51. We give strong preference to the oscillation by setting $d_p = 2.0$ in the preference term of Equation (7). The learning process automatically discovers the necessary transitions, which allows the character to land the board to regain balance when necessary and return to a stable oscillation.

We further test the learning process on a character with different mass distribution by attaching a heavy box of 10kg on the character's left arm (Figure 6(h)). This box is considered to be a part of the character when computing the features of the simulation such as the center of mass and the angular momentum. Without any modification, the control fragments and the scheduler learned for the normal character automatically allow this new character to balance on the bongo board several seconds before falling off the board, although out-of-sequence actions are frequently taken, and the character cannot maintain a stable oscillation. We then learn a new scheduler using the existing control fragments. As shown in Figure 7, this new scheduler consistently skips a few actions and executes the rest of the actions in sequence. The character thus maintains a more stable oscillation on the board at a higher frequency.

5.2.2 *Skateboarding.* We use the skateboard model shown in Figure 4(c), which is 0.8m long and weighs 2.8kg in total. It has four wheels of 5.6cm in diameter. A damping torque proportional to the rotational speed, $\tau = -0.001\omega$, is applied to each wheel. This torque slows down the skateboard so that the character has to push on the ground to keep moving forward. Similar to the bongo board balancing task, the inner-front end of the right foot is fixed on the board with a ball joint, which stabilizes the contacts while still allowing the foot to rotate. A side effect of this treatment is that the skateboard may be lifted unrealistically. To mitigate this problem, we augment the original construction process of the control fragments [Liu et al. 2016] with an extra cost term that penalizes any board lifting or tilting. This term effectively forces the control fragments to keep the skateboard on the ground. The environmental state $s_e = \{d_L, \theta_L, d_R, \theta_R\}$ is a six-dimensional vector containing the horizontal distance between the board and both of the character's feet, and their relative orientation around the vertical axis. The total DoF of the simulation state s is thus 24. The goal of the skateboarding task is to achieve interactive navigation on flat terrain. The user-controlled task parameter $U = \{\phi\}$ is the same as the one used in the walking-on-a-ball task. The task term E_{task} of the reward function is the direction cost of Equation (16) with $\epsilon_c = 2.0$m/s and $\epsilon_\phi = 10^\circ$.

Our control system learns the scheduler with an action set consisting of 39 control fragments. These control fragments are built from four reference motions, including pushing off the ground, rolling on the board, and two kick turns to the left and the right, respectively. The reference sequence O alternates pushing and the other movements. The action preference term $E_{\text{preference}}$ favors rolling on the board, which makes the character stay on the board as long as possible and push off the ground to accelerate if the board slows down too much. When the target direction changes, the turns are automatically activated when the character skateboards stably enough and executed repeatedly until the target direction is reached. Although the scheduler is learned on a flat terrain, it survives on a rough terrain with bumps of 2cm in height placed perpendicular to the skateboarding direction.

To show the effect of the preference term of the reward function, we test a second reward function where the subset of favored actions A_I includes the two kick turns instead of the rolling movement. A stronger preference parameter $d_p = 2.0$ is used for $E_{\text{preference}}$ in this experiment. The learning process finds a scheduler that alternates the two kick turns to accelerate, thus creating a *tic-tac* movement.

5.2.3 *Running.* The running task has the same goal as the skateboarding task and reuses the task parameter and the task reward term defined previously. Three running controllers are used for this task, including a forward run, a smooth right turn, and a 90° right turn. The mirrors of these turning controllers are also included to produce left turns. The reference sequence O randomly concatenated the forward run and the turns while keeping the foot contacts consistent. A total of 50 control fragments are included in the action set, among which the forward run is favored by the action preference term $E_{\text{perference}}$ of the reward function.

The scheduler learned by the proposed method automatically selects either the 90° turn or the smooth turn according to the difference between the current running direction and the target direction. Unlike the bongo boarding task and the skateboarding task, the learned scheduler for the running task follows the reference sequence O most of the time. In this sense, it behaves like a graph-based planner. To fully test the capability of the learned scheduler, we let the character (a) trip over a small bump on the ground and (b) run a few steps on a small patch of icy ground where the coefficient of friction is 0.1. In both situations, following the reference sequence leads to falling, but the learned scheduler can select out-of-sequence actions to prevent falling without any further adaptation, as shown in Figure 8. Furthermore, our control system can learn a new scheduler for the slippery terrain using the same set of control fragments. In this case, the character frequently uses out-of-sequence actions to maintain balance and slowly turns to the target direction.

5.2.4 *Push-Recovery.* In this task, we apply horizontal pushes to the character's trunk. The character responds by taking steps to

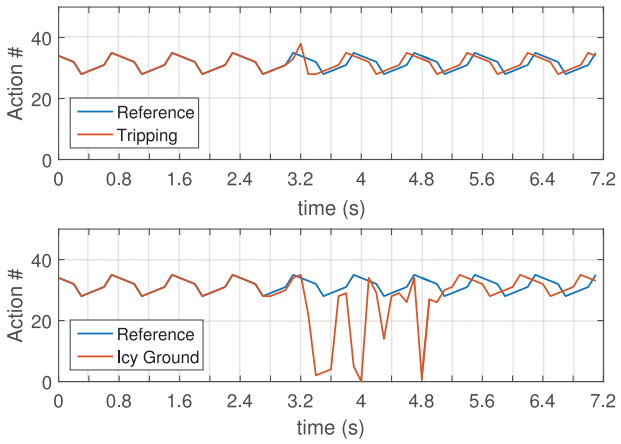


Fig. 8. Action sequences for the running task. Top: The character trips over a small bump. Bottom: The character runs a few steps on a patch of icy ground. The blue curves show the actions in the reference sequence. The red curves are the control fragments that are executed to the disturbances caused by the environment.

regain balance. Eight response movements are used for this task, each of which starts after a push from one of the eight directions and ends with the same standing pose after taking one or two steps. The reference sequence O contains only the transitions from these response movements to the in-place standing. There is no prior knowledge of which movement should be used to respond to a push. The scheduler is learned with an action set of 117 control fragments. The preference term $E_{\text{preference}}$ favors the standing actions, and the task term E_{task} is set to zero for this task. During the learning process, a random horizontal push of [100 N, 300 N] $\times$ 0.2s is applied on the character's trunk every 2s. The transition tuples obtained during these pushes are discarded. Using the learned scheduler, the character maintains in-place standing after moderate pushes of up to 70N $\times$ 0.2s from behind or 100N $\times$ 0.2s from the front and takes steps under the pushes in the range of [200N,300N] $\times$ 0.2s. Interestingly, our system can create forward and backward walks by applying a constant push of 40N from behind and from the front, respectively.

In addition to the control fragments constructed using the method proposed by Liu et al. [2016], our scheduling scheme can incorporate other types of control. To demonstrate the capability of our method, we replace all of the control fragments for standing in the push-recovery task with a QP-based (Quadratic Programming) balance controller and learn a new scheduler from this mixed action set of 109 control fragments. The new balance controller is implemented in the same way as described in Macchietto et al. [2009], except that the reference angular momentum is zero. The new scheduler checks the state of simulation every 0.1s and determines whether the character should take steps. We find that this balance controller is more robust than the feedback-augmented control fragments learned with Liu et al. [2016] in terms of maintaining in-place standing. As shown in the supplementary video, the character can tolerate a push of 100N$\times$0.2s from behind without stepping under the control of this new scheduler, while it has to step forward in the same situation with the old scheduler constructed with control fragments from Liu et al. [2016]. When a larger push is applied, the new scheduler still allows the character to take steps to regain balance.

5.2.5 *Breakdancing.* The character learns two breakdance stunts: a jump flip and a swipe movement. The user can interactively select one of the movements at runtime, and the character will try to perform the selected movement repeatedly until the user selection changes. We use two reference motion clips for this task, each containing a short preparation movement followed by one of the stunts. The action set includes 146 control fragments in total. The task parameter $U = \{A_0, A_1\}$ consists of two subsets of preferred actions, where A_0 contains the airborne phase of the jump flip, and A_1 contains the swipes of the swipe movement. The value of $u \in U$ is thus either 0 or 1. The task reward E_{task} is set to zero for this task. Instead, we give strong preference to the selected stunt by setting $d_p = 2.0$ in the preference term of Equation (7). The feedback threshold $\Delta_{\max}$ is set to 2.0 radians. Without any predefined transition, the learned scheduler responds to a user selection by finishing the current stunt, taking the preparation movement of the target stunt, and performing the new movement repeatedly.

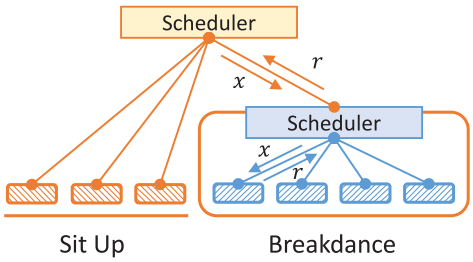


Fig. 9. The hierarchical scheduler that allows the breakdancer to recover immediately from a fall (Section 5.2.5). Each rounded rectangle represents a control fragment. The learned blue scheduler maintains the control fragments from the breakdance. It is considered as a special control fragment by the orange scheduler at the higher level. The orange scheduler also maintains a number of ordinary control fragments from the sitting up motion. At runtime, the orange scheduler picks a control fragment to execute according to the current state of the simulation, x. If the blue scheduler is selected, it automatically executes a blue control fragment according to x and returns the corresponding reward. The tracking penalty in the reward function (Equation (5)) is not used by the learned blue scheduler at this stage.

Both of the breakdance stunts are highly dynamic movements. Although the character can perform each movement tens of times under the control of the learned scheduler, we find that it occasionally fails and falls on the ground. After a fall, the character struggles for a few seconds before it is able to stand up and continue breakdancing. This sequence of actions is not natural looking. To mitigate this problem, we incorporate a sitting-up motion to help the character resume dancing quickly after falling. Specifically, we treat the scheduler learned earlier for breakdancing as a special control fragment and combine it with 35 ordinary control fragments from the sitting-up motion to create a hybrid action set. A new, hierarchical scheduler is learned using this hybrid action set, as shown in Figure 9, with the preference term of the reward function penalizing the sitting-up control fragments. With the new scheduler, the character can sit up immediately after falling and start to breakdance again.

5.3 Experiments on Implementation Choices

In this section, we review and test some of the implementation decisions that we made in learning these schedulers.

5.3.1 *State Vector.* Inspired by Liu et al. [2016], we choose a few high-level features to represent the state of the simulation. Our results indicate that this compact representation is applicable to a wide range of control tasks. However, our learning algorithm does not exclude the use of other state representations. For example, we

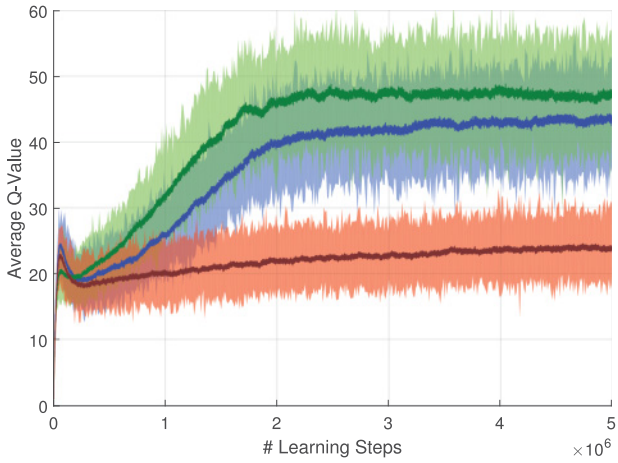


Fig. 10. Learning curves of the walking-on-a-ball task with different state vectors. Blue: Learning with the selected features as described in Section 4.1. Green: Learning with the positions and linear velocities of every rigid body as state vectors. Red: Learning without the environmental state.

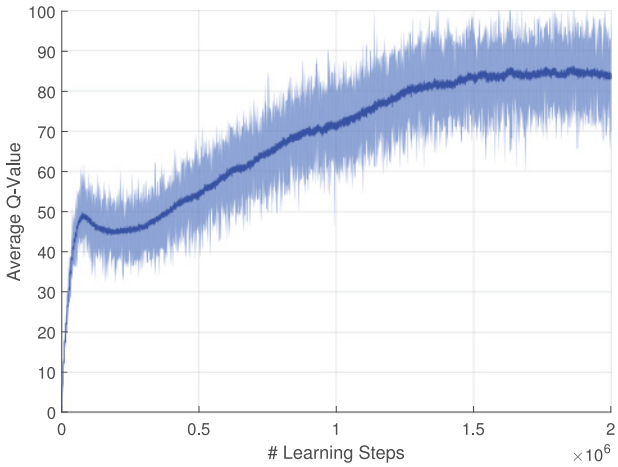


Fig. 11. Learning curve of the walking-on-a-ball task with delayed reward.

can learn successful schedulers for the ball walking task and the bango board balancing task by replacing the state vector, s_m, with a vector capturing the positions and linear velocities of all of the rigid bodies in the character model. Given that our character model consists of 20 rigid bodies, the dimension of the new state space S is 126, including the 6-DoF environmental state vectors, s_e, for each of the two tasks. Figure 10 shows the learning process for the ball walking task in this new state space using a green curve. Compared with the learning process using the compact state vector, this new learning process benefits from the additional information provided by the new state vector and converges at a higher average Q-value, indicating that the new scheduler takes less out-of-sequence actions than the original scheduler, although the learning process takes longer to reach such a scheduler due to the extra computational cost (Table I).

The environmental state s_e is important for tasks involving moving objects. For example, we cannot learn a successful scheduler for the ball walking task if s_e is excluded from the state vector. As shown in Figure 10 using a red curve, the learning process stops at a low average Q-value, and the character can only balance on the ball for 2s under the control of the learned scheduler.

5.3.2 *Immediate Reward vs. Delayed Reward.* Our learning method associates an immediate reward for every state during the learning process, which is an intuitive implementation choice for the tasks tested in this article. Without modifying the learning algorithm, we can also learn successful schedulers for the tasks that are defined with delayed rewards only given to the terminal states. To demonstrate this, we test a new ball walking task where the goal of the character is to stay on the ball as long as possible. The task parameter set U is empty for this task. Instead of using the immediate reward of Equation (5), we give a constant reward $R_0 = 5$ to every other state except the terminal states when the character falls off the ball, for which a penalty $R_{\text{fail}} = -5$ is applied. Figure 11 depicts the learning process for this special task. The character walks stably on the ball under the control of the learned scheduler, although the moving direction often changes because keeping that constant is not a part of the reward function.

6. DISCUSSION

In this article, we have demonstrated that the robustness of tracking control can be dramatically improved by reordering the control fragments when necessary, which enables the robust control of a wide range of behaviors and control tasks. Many of the behaviors tested in this article, such as walking on a ball, balancing on a bongo board, and skateboarding, cannot be robustly controlled by the previous time-indexed approach proposed by Liu and his colleagues [2016]. The instability and disturbances in dynamic environments cause these controllers to fail. In addition, the tasks such as push-recovery on the ground and on the bongo board cannot be accomplished without the scheduling algorithm.

We train our schedulers to maintain the reference timing of the input tracking controllers most of the time, which is crucial for the motion quality. Both the tracking penalty term of Equation (6) and the tracking exploration strategy used in the learning process play an important role in achieving this goal. For example, our system can learn a scheduler to accomplish the running task without the tracking penalty term, but the resultant forward run is quite jerky because the control fragments from the turns are often executed, given that they are similar to their counterparts from the forward run. The tracking penalty term provides an effective constraint in this situation to remove the ambiguity between similar control fragments and improve the motion quality.

The tracking exploration strategy effectively maintains a portion of in-sequence transition tuples in the sample set, which biases the training toward the reference sequence of the control fragments. This strategy may not be necessary when a task has a small action set, which is the case for the bongo board balancing task. As shown in Figure 3(a), using the random exploration strategy alone can eventually achieve a good enough result after more learning steps. However, for a task having a larger action set, such as skateboarding and running, the random exploration is too inefficient and ends in a local optimum as indicated in Figure 3. In this case, the out-of-sequence control fragments are often selected, degrading the motion quality.

Our method can only take actions from the predefined action set. This restriction limits the possible responses to disturbances. When the character encounters large perturbations, it may take an action that is not natural because there is no more appropriate action in the available action set. For example, when the character skateboards over bumps, it extends its left foot to keep balance. This motion is

actually a part of the push-off-the-ground action. A human skateboarder would more likely try to regain balance with a hip or ankle strategy executed with both feet on the board—an action that is not contained in the action set. This problem can be mitigated by adding missing control fragments into the action set and rerunning the learning process. Predicting the necessary missing actions using an active learning method such as Cooper et al. [2007] is an interesting topic for further investigation.

The action sets of the tasks presented in this article all consist of less than 150 control fragments. Our medium-sized Q-network structure that contains two 300-unit hidden layers works well for all of these tasks. We expect that incorporating more control fragments into a single action set will require a more complicated network structure, and the learning method will have to be adjusted to effectively train such networks.

When applicable, we always use a compact representation of the simulation state and a simple network structure to achieve fast ofline learning and online performance. The success of the tasks demonstrated in this article indicates that the selected features effectively capture the important information for a wide range of tasks. In addition, the experiments in Section 5.3.1 demonstrate that our medium-sized network also works well with the high-dimensional state representation that captures the raw states of the rigid bodies. The extra information can help the learning algorithm find a better scheduler, although learning time increases and performance degrades because of the computation caused by the extra dimensions. We expect that such a simple network structure can be generalized well to other tasks which can be described using only the information of the character. For the tasks where the character interacts with a complex environment, such as running over uneven terrain [Peng et al. 2016], fully utilizing the capability of deep learning algorithms and directly extracting high-level features of the environment will be crucial to the learning of successful control policies. We would like to explore in this direction in future research.

Switching to an out-of-sequence control fragment often causes discontinuous control signals. The physics of the simulation smooths out such discontinuities but may produce abrupt movement when the gap between two successive control fragments is too large. This problem partially explains the necessity of referring to the original timing. Blending the control from multiple control fragments with a kinematics-based method such as Lee et al. [2010b] is a possible way to solve this problem and to achieve rapid response to user interaction, although a specific interpolation method such as the one investigated in da Silva et al. [2009] may be needed.

When applicable, the basins of attraction of the feedback policies of all the control fragments can be viewed as a discretization of the state space near the reference motion. From this perspective, learning a scheduler is equivalent to training a nonlinear classifier that implicitly determines the decision boundary. Traditional machine learning algorithms, such as the Support Vector Machine (SVM) and the k-nearest neighbor algorithm, may also be used to solve the problem, although obtaining well-labeled training data and tuning the distance metric may be challenging.

Our schedulers pick an action every 0.1s at runtime, which is a simple implementation choice suggested by Liu et al. [2016] and is shown to be effective for all the tasks discussed in this article. In practice, this 0.1-s interval makes the input states to a scheduler separate well, as shown in Figure 12, and thus facilitates the learning of the schedulers. If more information is available, the segmentation of a motion may be further improved by taking into account the critical moments such as when the character's feet or hands contact the ground in a backflip. In future research, we are also interested in learning the schedulers that plan the duration of actions as well,

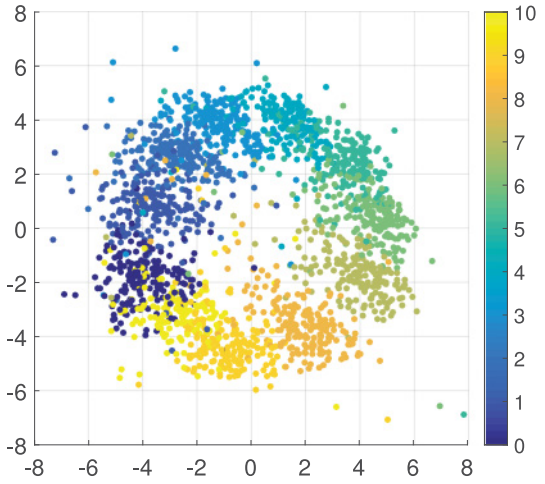


Fig. 12. The first two principal components of the states input to the learned scheduler in the bongo board balancing task. The color of each data point represents the action taken at the corresponding state.

which can be seen as a special case of the Semi-Markov Decision Process [Sutton et al. 1998].

A number of recent works also learn neural-network-based control policies that directly map a state to continuous control signals for actuators [Levine and Koltun 2013, 2014; Tan et al. 2014; Mordatch et al. 2015]. The continuous action spaces necessitate the use of policy search or policy optimization. The control fragments used in our system can be viewed as an additional layer of the Q-network. Such a combined neural network is equivalent to the networks learned by those methods. In this sense, we divide a hard control problem into two subproblems, each of which is easier to solve than the combined problem.

ACKNOWLEDGMENTS

We would like to thank all the anonymous reviewers for their valuable comments and suggestions. We thank Stelian Coros for the discussions in the early stage of this project. We thank Moshe Mahler and Kyna McIntosh for their help in creating the final demo.

REFERENCES

Yeuhi Abe and Jovan Popovíc. 2011. Simulating 2D gaits with a phase-indexed tracking controller. *IEEE Comput. Graph. Appl.* 31, 4 (July 2011), 22–33.

Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. 2013. Trajectory optimization for full-body movements with complex contacts. *IEEE Trans. Visual. Comput. Graph.* 19, 8 (Aug 2013), 1405–1414.

Mazen Al Borno, Eugene Fiume, Aaron. Hertzmann, and Martin de Lasa. 2014. Feedback control for rotational movements in feature space. *Comput. Graph. Forum* 33, 2 (May 2014), 225–233.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: New features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: A CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Brian G. Buss, Alireza Ramezani, Kaveh Akbari Hamed, Brent A. Griffin, Kevin S. Galloway, and Jessy W. Grizzle. 2014. Preliminary walking experiments with underactuated 3D bipedal robot MARLO. In *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*. 2529–2536.

Seth Cooper, Aaron Hertzmann, and Zoran Popović. 2007. Active learning for real-time motion controllers. *ACM Trans. Graph.* 26, 3 (July 2007), Article 5.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust task-based control policies for physics-based characters. *ACM Trans. Graph.* 28, 5 (Dec. 2009), Article 170, 170:1–170:9.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized biped walking control. *ACM Trans. Graph.* 29, 4 (July 2010), Article 130, 130:1–130:9 pages.

Marco da Silva, Frédo Durand, and Jovan Popović. 2009. Linear Bellman combination for control of character animation. *ACM Trans. Graph.* 28, 3 (July 2009), Article 82, 82:1–82:10.

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive character animation using simulated physics: A state-of-the-art review. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2492–2515.

Gaël Guennebaud, Benoît Jacob, and others. 2010. Eigen v3. Retrieved from http://eigen.tuxfamily.org.

Sehoon Ha and C. Karen Liu. 2014. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Trans. Graph.* 34, 1, Article 1 (Dec. 2014), 1:1–1:11.

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), 81:1–81:13.

Sumit Jain and C. Karen Liu. 2011. Controlling physics-based characters using soft contacts. *ACM Trans. Graph.* 30, 6 (Dec. 2011), Article 163, 163:1–163:10.

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)*. 129–138.

Taesoo Kwon and Jessica K. Hodgins. 2017. Momentum-mapped inverted pendulum models for controlling dynamic human motions. *ACM Trans. Graph.* 36, 1, Article 10 (Jan. 2017), 14 pages.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010a. Data-driven biped control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 129:1–129:8.

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010b. Motion fields for interactive character locomotion. *ACM Trans. Graph.* 29, 6, Article 138 (Dec. 2010), 138:1–138:8.

Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*.

Sergey Levine and Vladlen Koltun. 2014. Learning complex neural network policies with trajectory optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015). http://arxiv.org/abs/1509.02971

Chenggang Liu, Christopher G. Atkeson, and Jianbo Su. 2013. Biped walking control using a trajectory library. *Robotica* 31, 2 (March 2013), 311–322.

Libin Liu, Michiel van de Panne, and KangKang Yin. 2016. Guided learning of control graphs for physics-based characters. *ACM Trans. Graph.* 35, 3 (May 2016), Article 29, 29:1–29:14.

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving sampling-based motion control. *Comput. Graph. Forum* 34, 2 (2015), 415–423.

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 (2010), Article 128.

Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. 2013. Simulation and control of skeleton-driven soft body characters. *ACM Trans. Graph.* 32, 6 (2013), Article 215.

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum control for balance. *ACM Trans. Graph.* 28, 3 (2009).

James McCann and Nancy Pollard. 2007. Responsive characters from motion fragments. *ACM Trans. Graph.* 26, 3 (July 2007), Article 6.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 Feb 2015), 529–533. http://dx.doi.org/10.1038/nature14236. Letter.

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.* 29, 4 (July 2010), Article 71, 71:1–71:8.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V. Todorov. 2015. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 3114–3122.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.* 31, 4 (July 2012), Article 43, 43:1–43:8.

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.* 28, 3 (2009).

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. 2015. Massively parallel methods for deep reinforcement learning. In *Deep Learning Workshop, International Conference on Machine Learning (ICML'15)*.

Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. 2004. Learning from demonstration and adaptation of biped locomotion. *Robot. Auton. Syst.* 47, 23 (2004), 79–91.

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic terrain traversal skills using reinforcement learning. *ACM Trans. Graph.* 34, 4, Article 80 (July 2015), 80:1–80:11.

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Trans. Graph.* 35, 4 (July 2016).

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

Freek Stulp, Evangelos A. Theodorou, and Stefan Schaal. 2012. Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Trans. Robot.* 28, 6 (Dec 2012), 1360–1370.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. 1998. Intra-option learning about temporally abstract actions. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*. 556–564.

Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. 2014. Learning bicycle stunts. *ACM Trans. Graph.* 33, 4 (July 2014), Article 50, 50:1–50:12.

Jie Tan, C. Karen Liu, and Greg Turk. 2011. Stable proportional-derivative controllers. *IEEE Comput. Graph. Appl.* 31, 4 (2011), 34–44.

Tijmen Tieleman and Geoff Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

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

Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep reinforcement learning with double Q-learning. *CoRR* abs/1509.06461 (2015). http://arxiv.org/abs/1509.06461

Kevin Wampler and Zoran Popović. 2009. Optimal gait and form for animal locomotion. *ACM Trans. Graph.* 28, 3 (2009), Article 60.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2010. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.* 29, 4, Article 73 (July 2010), 73:1–73:8.

Christopher John Cornish Hellaby Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. dissertation. King's College, Cambridge, UK. http://www.cs.rhul.ac.uk/ chrisw/new_thesis.pdf.

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), 74:1–74:9.

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

Victor Zordan, David Brown, Adriano Macchietto, and KangKang Yin. 2014. Control of rotational dynamics for ground and aerial behavior. *IEEE Trans. Visual. Comput. Graph.* 20, 10 (Oct. 2014), 1356–1366.

Received September 2016; revised February 2017; accepted March 2017