Guided Learning of Control Graphs for Physics-based Characters

LIBIN LIU and MICHIEL VAN DE PANNE
The University of British Columbia
and
KANGKANG YIN
National University of Singapore

The difficulty of developing control strategies has been a primary bottleneck in the adoption of physics-based simulations of human motion. We present a method for learning robust feedback strategies around given motion capture clips as well as the transition paths between clips. The output is a control graph that supports real-time physics-based simulation of multiple characters, each capable of a diverse range of robust movement skills, such as walking, running, sharp turns, cartwheels, spin-kicks, and flips. The control fragments which comprise the control graph are developed using guided learning. This leverages the results of open-loop sampling-based reconstruction in order to produce state-action pairs which are then transformed into a linear feedback policy for each control fragment using linear regression. Our synthesis framework allows for the development of robust controllers with a minimal amount of prior knowledge.

1. INTRODUCTION

Designing controllers to realize complex human movements remains a challenge for physics-based character animation. Difficulties arise from non-linear dynamics, an under-actuated system, and the obscure nature of human control strategies. There is also a need to design and control effective *transitions* between motions in addition to the individual motions. Since the early work on this problem over two decades ago, controllers had been developed for many simulated skills, including walking, running, swimming, numerous aerial maneuvres, and bicycle riding. However, controller design often relies on specific insights into the particular motion being controlled, and the methods often do not generalize for wider classes of motions. It also remains difficult to integrate motion controllers together in order to produce a multi-skilled simulated character.

In this paper, we develop controllers for a wide variety of realistic, dynamic motions, including walking, running, aggressive turns, dancing, flips, cartwheels, and getting up after falls, as well as transitions between many of these motions. Multiple simulated characters can physically interact in real-time, opening the door to the possible use of physics in a variety of sports scenarios.

Our method is designed around the use of motion capture clips as reference motions for the control, which allows for existing motion capture data to be readily repurposed to our dynamic setting. It also helps achieve a high degree of realism for the final motion without needing to experiment with objective functions and solution shaping, as is often required by optimization approaches. The control itself is broken into a sequence of *control fragments*, each typically $0.1s$ in length, and a separate linear feedback control strategy is learned for each such fragment. An iterative *guided learning* process is used for learning: a sampling-based control method serves as a control oracle that provides high-quality solutions in the form of state-acion pairs; linear regression on these pairs then provides an estimated linear control policy for any given control fragment. Importantly, successive iterations of the learning are coupled together by using the current estimated linear control policy to inform the construction of the solution provided by the control oracle; it provides the oracle with an estimated solution, which can then be refined as needed. This coupling encourages the oracle and the learned control policy to produce mutually compatable solutions. The final control policies are compact in nature and have low computational requirements.

Our work makes two principal contributions: (1) A guided-learning algorithm that combines the use of a sampling-based control oracle and the full-rank linear regression for iteratively learning time-varying linear feedback policies that robustly track input motion capture clips. The pipeline further supports motion retargeting. (2) An overall clips-to-controllers framework that learns robust controllers for a wide range of cyclic and non-cyclic human motions, including many highly dynamic motions, as well as learning transitions between the controllers in order to produce flexible control graphs. Results demonstrate the integrated motion capabilities on real-time simulations of multiple characters that are capable of physics-based interactions with each other. Four different stand-up strategies, each based on motion capture data, allow characters to recover from falls voluntarily.

2. SYSTEM OVERVIEW

Figure 1 provides an overview of the system components and how they interact. As input, the system takes individual motion clips and the desired connectivity of these clips, as represented by a motion graph. The output is then a control graph that is constructed from a large set of *control fragments*, typically of short duration, e.g., $0.1s$, which afford the connectivity described by the desired motion graph. Each control fragment is defined by a short target tracking trajectory, $\hat{\boldsymbol{m}}$, its duration, δt, and a related linear feedback policy, $\boldsymbol{\pi}$, as developed during an offline *guided learning* process.

The learning process begins with the application of a SAMpling-based CONtrol strategy (SAMCON) that produces an open-loop trajectory for the controls, and therefore each control fragment, that does well at reproducing a given input motion clip or motion-clip transition. This serves two purposes. First, it replaces the input motion, which is often not physically feasible due to modeling errors and possible retargeting, with a physically-realizable motion. Second, it provides a nominal open-loop reference motion and the associated control values, around which we will then learn linear feedback control strategies to provide robust control. The reference

{libinliu, van}@cs.ubc.ca, kkyin@comp.nus.edu.sg

motion and the control values are stored in the control fragments that underly any given motion clip.

Next, iterative guided learning is used to learn linear feedback policies for each control fragment. This involves the repeated use of SAMCON in order to produce multiple new solutions (motions and the control values underlying them) that each do well at reproducing the reference motion. These then serve to provide state-and-corresponding-action data for learning a local linear feedback model for each control fragment, using linear regression. However, the initial linear feedback policies learned in this fashion do not work well in practice; when applied in simulation, the resulting motion quickly visits regions of the state-space that are far-removed from the regions for which the original state-and-action data was obtained. To remedy this, *guided SAMCON* uses the current linear control policy as an initial guess for computing its solutions, thereby implicitly looking for control solutions that exhibit a degree of compatibility with the current linear-feedback control policies. Over multiple iterations of the guided learning loop, the process converges towards robust linear feedback control policies for the sequence of control fragments. In order for the described method to also be able to robustly handle transitions between motion clips, as modeled by the desired connectivity in the motion graph, the motions for which we use SAMCON to collect the desired state-and-action data will come from long random walks on the desired motion graph. In this way, a control fragment that immediately follows incoming transitions from multiple branches of a motion graph will see state-and-action data from all of these different arrival paths and will therefore be encouraged to produce a control policy that is compatible with this possibly-diverse set of starting states.

During online simulation, a motion planner or user input specifies a desired path through the control graph, and thus provides a desired sequence of control fragments. Linear feedback control is applied once at the beginning of each control fragment based on the current state at that time. The computed control action specifies a constant offset from the reference controls that is then applied for the duration of the control fragment. The linear feedback control decisions are thus made at the time scale of the control fragments. Finally, proportional-derivative (PD) controllers are used to convert the control actions, which are target joint angles in our case, into joint torques. The use of this final low-level control construct at a fine time scale (milliseconds) allows for the rapid, adaptive generation of torques upon ground contact or other collisions, and helps enable the key control decisions to be made at the coarser time scale of the control fragments.

3. RELATED WORK

Numerous approaches have been proposed for controlling the motion of physics-based characters, many of which are described in a recent survey article [Geijtenbeek and Pronost 2012]. In what follows below, we review various approaches and categorize them according to the features of relevance to our proposed method. Suitably crafted phase structures, abstractions of state-and-action features, and the use of optimization are important common features across a majority of approaches.

Implicit dynamics methods:

Many control strategies have been developed that do not require the controller to have full knowledge of the equations of motion. Instead, these are taken into account implicitly during multiple simulations that are used to evaluate the impact of parameter adaptations that are made to the components of the control system. These methods are commonly characterized by proportional-derivative joint

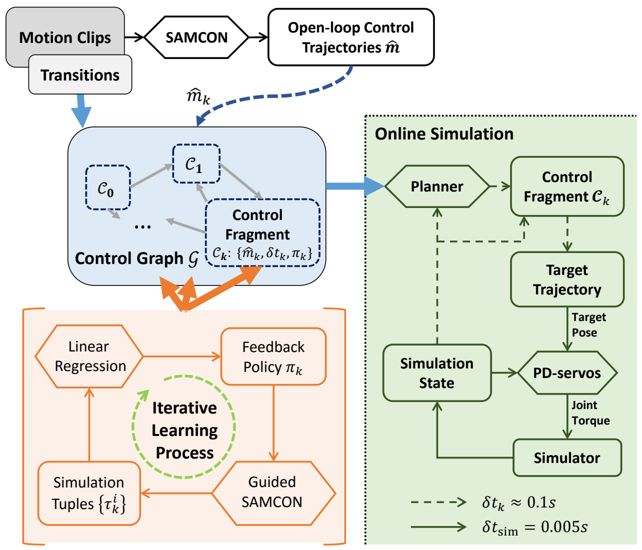

Motion Clips
Transitions
SAMCON
Open-loop Control Trajectories $\hat{m}$
$\hat{m}_k$
$\mathcal{C}_0$
$\mathcal{C}_1$
...
Control Fragment
$\mathcal{C}_k$: $\{\hat{m}_k, \delta t_k, \pi_k\}$
Control Graph $\mathcal{G}$
Linear Regression
Feedback Policy π_k
Iterative Learning Process
Simulation Tuples $\{\tau_k^i\}$
Guided SAMCON
Online Simulation
Planner
Control Fragment $\mathcal{C}_k$
Target Trajectory
Target Pose
Simulation State
PD-servos
Joint Torque
Simulator
$\delta t_k \approx 0.1s$
$\delta t_{\text{sim}} = 0.005s$

Fig. 1: System Overview

control, force generation using the Jacobian tranpose, finite state machines that model motion phases, and phase-specific feedback laws that govern tasks such as hand and foot placement. Controllers for many types of motion skills have been synthesized, including walking, running, bicycling, and other agile motions, e.g., [Hodgins et al. 1995; Yin et al. 2007; Wang et al. 2009; Kwon and Hodgins 2010; Lee et al. 2010; Ha et al. 2012; Liu et al. 2012; Al Borno et al. 2013; Tan et al. 2014]. Knowledge and insights about the desired motions can be incorporated into the design of the control system, an objective function to be used for optimization, or, most commonly, using both of these. Realistic muscle models can also be integrated into such approaches, i.e., [Wang et al. 2012; Geijtenbeek et al. 2013].

Optimized inverse dynamics methods: Another popular category of approach combines knowledge of the equations of motion with optimization in order to solve directly for the control actions, typically joint torques. This can be done at various time scales. Short time-horizon methods optimize for the controls, accelerations, and ground contact forces for the current time step and are commonly solved using quadratic programming, which allows for ground contact constraints to be conveniently imposed. Knowledge about the phase-based nature of the motion can be encoded into phase-specific objective functions, and anticipatory knowledge can be incorporated into simplified models that then participate in the objective function. The approach has been successfully applied to a wide range of motions, e.g., [Da Silva et al. 2008; Macchietto et al. 2009; de Lasa et al. 2010; Ye and Liu 2010; Zordan et al. 2014; Al Borno et al. 2014]. Long-horizon methods optimize for the motion and the underlying controls for a finite-duration horizon into the future, possibly encompassing the entire motion, e.g., [Popović and Witkin 1999; Sulejmanpašić and Popović 2005; Wampler and Popović 2009]. For interactive applications, model-predictive control is used, whereby only the immediate control actions are employed and the remainder of the time horizon is treated as a motion plan that is then extended and reoptimized at the next control time step, e.g., [Tassa et al. 2012]. Recent work has further shown that the phase structure can also be learned for a variety of motions [Mordatch et al. 2012].

Motion tracking: Motion capture data can be used as part of controller design as a means of producing high-quality motions without needing to first fully decipher the many factors that may influence how humans move. Motion capture clips had been used as reference trajectories for passive simulation [Zordan et al. 2005] and spacetime optimization [Popović and Witkin 1999; Sulejmanpašić and Popović 2005]. With the help of robust abstract feedback policies, it can be used to guide the creation of closed-loop controllers for realistic walking [Sok et al. 2007; Yin et al. 2007; Lee et al. 2010] and running [Kwon and Hodgins 2010] motions. Model-based optimal control provides a general method for developing robust control about given reference trajectories [Muico et al. 2009; Muico et al. 2011]. In general, however, it remains unclear how to adapt tracking-based control methods for complex contact conditions and for a wide range of motions. The sampling-based control strategy proposed in [Liu et al. 2010] has demonstrated the ability to robustly track a wide variety of motions, including those involving complex changing contacts. However, the solutions are open-loop and require offline computation.

Compact linear feedback: Low-dimensional linear feedback policies can performs surprisingly well in many circumstances, suggesting that compact and simple solutions do often exist for producing robust control for locomotion [Raibert and Hodgins 1991; Yin et al. 2007]. Robust reduced-order linear feedback policies can also be learned for a variety of motions using optimization in the space of reduced-order linear policies [Ding et al. 2015]. This method has further been demonstrated in the synthesis of control for several parkour-like skills [Liu et al. 2012] and skeleton-driven soft body characters [Liu et al. 2013]. However, using the same reduced-order linear policy across all phases of a motion is insufficient for complex motions, and thus the work of Liu et al. [2012] requires manually segmentation into motion phases, followed by the optimization of separate feedback policies for each motion phase. In this paper, we avoid the need for this manual segmentation by allowing each short-duration control fragment to have its own linear feedback model for its fine-scale (approximately $0.1s$) motion phase. Our regression-based learning can efficiently learn the large number of linear feedback parameters that result from the resulting parameter-rich model.

Multiple controller integration: Kinematic approaches offer easy-to-use graph structures for organizing and composing motion clips. However, research on sequencing and interpolation of controllers remains sparse. Given a set of existing controllers, oracles can be learned to predict the basins of attraction for controllers, and therefore to predict when transitions can safely be made between controllers [Faloutsos et al. 2001]. Tracking multiple trajectories simultaneously has been used to enhance the robustness of locomotion control [Muico et al. 2011]. Transitions between running and obstacle clearing maneuvers are realized in [Liu et al. 2012] using careful design of the structure and objectives of the transition behaviors. In this paper, we systematically realize robust transitions between many different skills.

Reinforcement learning: Reinforcement learning (RL) provides a convenient and well-studied framework for control and planning. It seeks an optimal policy that maximizes the expected returns given rewards that characterize a desired task. Value-iteration RL methods have been used on kinematic motion models, e.g., for boxing[Lee and Lee 2006] and flexible navigation [Lee and Lee 2006; Treuille et al. 2007; Lee et al. 2010], and for physics-based models, e.g., terrain traversal with constraints [Coros et al. 2009] and with highly dynamic gaits [Peng et al. 2015]. Policy search methods are often applied to problems having continuous action spaces, often searching the parameter space using stochastic optimization algorithms such as policy gradient [Peters and Schaal 2008], related EM-based approaches [Peters and Schaal 2007], and approaches with compact-but-adaptive policy representations [Tan et al. 2014]. Despite such progress, policy search often suffers from common issues related to optimization in high-dimensional spaces, such as being sensitive to the policy representation, requiring large number of samples, and convergence to local optima. Several recent works make progress on this problem using forms of *guided policy search*, an iterative process where new samples from a control oracle inform the construction of an improved policy, which then informs the collection of new samples, and so forth, e.g., [Ross et al. 2011; Levine and Koltun 2013; 2014; Mordatch and Todorov 2014].

Symbol	Description
$\boldsymbol{p}$	pose
$\hat{\boldsymbol{p}}$	target pose for PD-servos
$\Delta\hat{\boldsymbol{p}}$	offset on target poses
$\boldsymbol{m}$	motion clip, i.e. a sequence of poses in time
$\tilde{\boldsymbol{m}}$	reference motion capture clip
$\hat{\boldsymbol{m}}$	control clip / tracking target trajectory
$\tilde{\mathcal{G}}$	reference motion graph
$\mathcal{G}$	control graph
$\mathcal{C}$	control fragment
δt	duration of a control fragment
$\boldsymbol{\pi}$	feedback policy of a control fragment
$\boldsymbol{M}, \hat{\boldsymbol{a}}$	gain matrix and affine term of a feedback policy
$\boldsymbol{\Sigma}$	variance of policy explorations
$\boldsymbol{s}$	state vector
$\boldsymbol{a}$	action vector
τ_k	simulation tuple corresponds to $\mathcal{C}_k$. $\tau_k = (\boldsymbol{s}_{k-1}, \boldsymbol{a}_k, \boldsymbol{s}_k)$
$\mathcal{W}$	random walk on the control graph $\mathcal{W} = \{\mathcal{C}_k\}$
$\boldsymbol{\tau}$	execution episode of the random walk, $\boldsymbol{\tau} = \{\tau_k\}$
i	sample index for policy search
j	sample index for guided SAMCON
k	index for control fragments

Table I. : Symbols

Our learning pipeline has a similar guided-learning structure but is unique in its use of: (1) the use of an implicit-dynamics, sample-based motion reconstruction method as the control oracle; (2) the use of simple time-indexed linear feedback policies and linear regression to learn these policies; (3) a focus on difficult, dynamic, and realistic 3D full-body human motion skills; and (4) the ability to learn transitions between skills to yield integrated multiskilled characters.

4. STRUCTURE OF CONTROLLERS

We model a virtual character as an under-actuated articulated rigid body system, whose pose $\boldsymbol{p} = (\boldsymbol{x}_0, \boldsymbol{q}_0, \boldsymbol{q}_j), j = 1, \ldots, n$ is fully determined by the position ($\boldsymbol{x}_0$) and orientation ($\boldsymbol{q}_0$) of the root and the rotations of all n joints. We drive each DoF (degree of freedom) with PD-servos:

$$\tau = k_p(\hat{q} - q) - k_d\dot{q} \tag{1}$$

where q and $\dot{q}$ represent the joint rotation and rotational speed respectively, and the tracking target $\hat{q}$ is given by a target pose $\hat{\boldsymbol{p}}$. The system is simulated with the open-source Open Dynamic Engine (ODE). For better stability, we follow the idea of Stable-PD control [Tan et al. 2011] and replace the second term of Equation 1 with implicit damping in the same way as described in [Liu et al. 2013]. This allows us to use a large simulation time step (5ms),

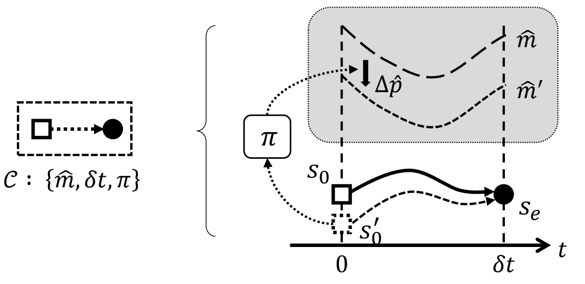


Fig. 2: A control fragment: when the simulate state $\boldsymbol{s}_0'$ drifts away from the reference start state $\boldsymbol{s}_0$, the feedback policy $\boldsymbol{\pi}$ is involved to compute a compensation $\Delta\boldsymbol{p}$ that offsets the open-loop control clip $\hat{\boldsymbol{m}}$ to $\hat{\boldsymbol{m}}'$. By tracking $\hat{\boldsymbol{m}}'$ with PD-servos, the simulation can end near the reference end state $\boldsymbol{s}_e$ in δt seconds.

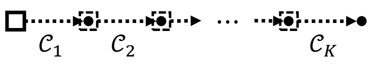


Fig. 3: A chain of control fragments

which significantly speeds up the learning process and improves the online performance.

Control fragments, represented by the symbol $\mathcal{C}$, are the basic units of the controllers in our framework. A control fragment is a tuple $\{\delta t, \hat{\boldsymbol{m}}, \boldsymbol{\pi}\}$ as indicated in Figure 2, where $\hat{\boldsymbol{m}} = \hat{\boldsymbol{p}}(t)$ represents an open-loop control clip consists of a sequence of target poses in time, which can be tracked by PD-servos to simulate a character from a start state $\boldsymbol{s}_0$ to the end state $\boldsymbol{s}_e$ in δt seconds. $\boldsymbol{s}_0$ and $\boldsymbol{s}_e$ are derived from the reference $\hat{\boldsymbol{m}}$. In practice, the simulation state in effect when a control fragment begins, $\boldsymbol{s}_0'$, will not be exactly at the expected starting state, $\boldsymbol{s}_0$, due to perturbations. The feedback policy, $\boldsymbol{\pi}$, is therefore used to compute a corrective action, a, which consists of an offset, $\Delta\hat{\boldsymbol{p}}$, that is added to the $\hat{\boldsymbol{m}}$ in order to eliminate the deviation. As illustrated in Figure 2, this offset remains fixed during the whole control fragment, yielding a resulting control clip $\hat{\boldsymbol{m}}' = \Delta\hat{\boldsymbol{p}} \oplus \hat{\boldsymbol{m}}$ that is then tracked instead of $\hat{\boldsymbol{m}}$ in order to have the state end nearby the desired end state, $\boldsymbol{s}_e$. Here the operator $\oplus$ represents a collection of quaternion multiplications between corresponding joint rotations.

Our framework employs a linear feedback policy for every control fragment:

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{\pi}(\boldsymbol{s}; \boldsymbol{M}, \hat{\boldsymbol{a}}) \\ &= \boldsymbol{M}\boldsymbol{s} + \hat{\boldsymbol{a}} \end{aligned} \tag{2}$$

where $\boldsymbol{M}$ represents a feedback gain matrix, $\hat{\boldsymbol{a}}$ is an affine term, and $\boldsymbol{s}$ and $\boldsymbol{a}$ are vectors representing the simulation state and feedback action, respectively. We use a selected subset of state and action features in order to facilitate a compact control policy. For all the skills developed in this paper, we use $\boldsymbol{s} = (\boldsymbol{q}_0^*, h_0, \boldsymbol{c}, \dot{\boldsymbol{c}}, \boldsymbol{d_l}, \boldsymbol{d_r}, \boldsymbol{L})$, consisting of the root orientation $\boldsymbol{q}_0^*$, the root height h_0, the centroid position $\boldsymbol{c}$ and velocity $\dot{\boldsymbol{c}}$, vectors pointing from the center of mass to the centers of both feet $\boldsymbol{d_l}, \boldsymbol{d_r}$, and the angular momentum $\boldsymbol{L}$. All these quantities are measured in a coordinate frame that has one axis vertically aligned and another aligned with the character's facing direction. As the vertical component of the root orientation $\boldsymbol{q}_0$ is always zero in this reference frame, $\boldsymbol{q}_0^*$ contains only the two planar components of the corresponding exponential map of $\boldsymbol{q}_0$. $\boldsymbol{s}$ thus represents 18 degrees of freedom (DoF). Similarly, we use an 11-DoF action vector $\boldsymbol{a}$ that consists of the offset rotations of the waist, hips, and knees, represented in terms of exponential map. Knee joints have one DoF in our model. The final compensation offset, $\Delta\hat{\boldsymbol{p}}$, is then computed from $\boldsymbol{a}$, where we set the offset rotations of all remaining joints to zero.

A controller can be defined as a cascade of control fragments, as depicted in Figure 3, which can be executed to reproduce a given motion clip. We also wish to be able to organize the control fragments into a graph as shown in Figure 4(b), whereby multiple possible outgoing or incoming transitions are allowed at the boundaries of the control fragments at *transition states*, such as $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$. We further define the chains of the control fragments between transition states as *controllers* and each controller is uniquely colored in Figure 4(b). In practice, controllers need to produce particular skills, e.g., running, and to perform dedicated transitions between skills, e.g. speeding up to a run. In Figure 4(c) we then illustrate the corresponding connectivity between controllers. Here, an arrow indicates that the controller associated with the *tail* ends in a state that is near to the expected starting state of the controller associated with the *head*. Based on this graph structure, the sequencing of skills is simply achieved by walking on this graph while executing the encountered control fragments.

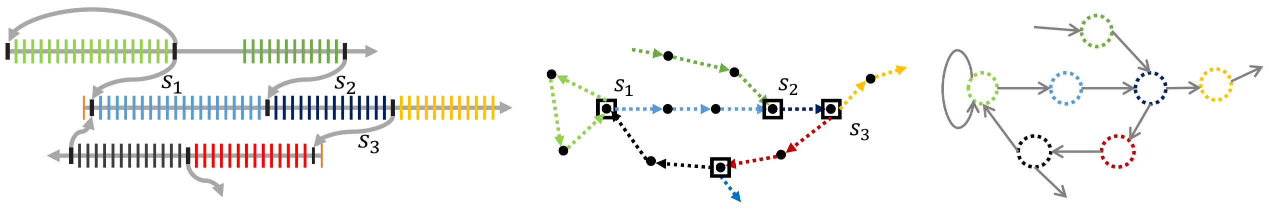
In our framework, the structure of a control graph is predefined and fixed during the learning process. Given example motion clips of desired skills, this is done by first building a reference motion graph, and then converting it into a control graph. Figure 4(a) shows a simple motion graph consisting of three motion clips and transitions between sufficiently similar frames, e.g. $\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3$, which define the transition states. Any portion of a motion clip that is between two transition frames is then converted to a chain of control fragments, or equivalently, a controller, between the corresponding transition states. In this conversion, the motion clip is segmented into K identical-duration pieces, with K chosen to yield time intervals $\delta t \approx 0.1s$. We construct high-quality open-loop control trajectories from the input motion clips using the improved SAMCON algorithm and noise reduction and time scaling techniques [Liu et al. 2015; Liu et al. 2013], and initialize the control fragments with the resulting open-loop controls. The feedback policies $\boldsymbol{\pi}$ are initialized to zero, i.e. $\boldsymbol{M} = \boldsymbol{0}, \hat{\boldsymbol{a}} = \boldsymbol{0}$.

The initial configuration of control fragements as described thus far cannot produce robust execution of skills because of the lack of feedback. In the next section, we introduce the details of our learning pipeline that augment the control graph with a feedback policy for each control fragment.

5. GUIDED LEARNING OF FEEDBACK POLICIES

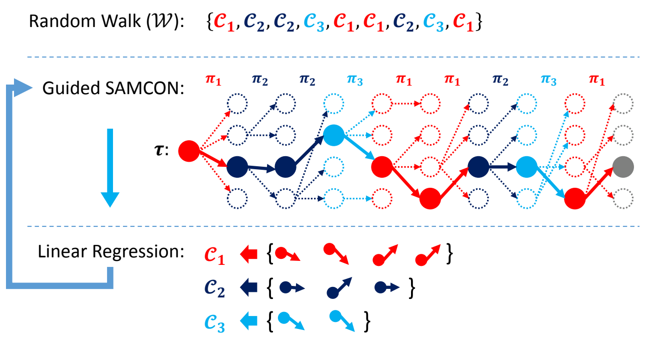
We desire a control graph that supports random walks on the graph for physics-based characters, analogous to the use of a motion graph for kinematic motion synthesis. For these physics-based skills to be robust, feedback policies need to be developed for the control fragments. Formally, given a control graph that consists of K control fragments, $\{\mathcal{C}_k\}$, we need to learn feedback policies for these control fragments that ensure successful random walks $\mathcal{W} = \{\mathcal{C}_{k_1}, \mathcal{C}_{k_2}, \dots\}, k_i \in \{1, \dots K\}$ on the graph. To this end, we first generate a long sequence $\mathcal{W}$ via a random walk on the control graph, in which each control fragment $\mathcal{C}_k$ appears at least 200 times. We then formulate the learning process as a policy search problem to be evaluated on $\mathcal{W}$, and use an iteratively process to develop suitable feedback policies.

Figure 5 provides a toy illustration of the guided learning process. Given a random graph-walk, $\mathcal{W}$, consisting of 9 control fragments, a successful execution episode of $\mathcal{W}$ is generated using



(a) a motion graph (b) a control graph (c) a compact representation of (b)

Fig. 4: Control graph: a control graph is created by (a) building a reference motion graph from example motion clips, then (b) converting each clip of the motion graph to a chain of control fragments. (c) shows a compact representation of the control graph (b), where each node represent a chain of control fragments, or rather, a controller.



Random Walk ($\mathcal{W}$): $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_1, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_1\}$
Guided SAMCON:
π_1 π_2 π_2 π_3 π_1 π_1 π_2 π_3 π_1
τ:
Linear Regression:
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{C}_3$

Fig. 5: A sketch of the guided learning process for a toy control graph.

Guided SAMCON, as will be discussed in further detail in §5.3. This provides a sequence, $\boldsymbol{\tau}$, of states and corresponding control actions that does well at reproducing the desired reference motions corresponding to $\mathcal{W}$. In this toy example, we simply use four discrete states as an abstract representation of a larger continuous state space, and the actions are simply represented as the arrows that transition to the state at the start of the next control fragment. Because each control fragment occurs multiple times in $\mathcal{W}$, multiple state-action pairs, $(\boldsymbol{s}, \boldsymbol{a})$, are collected for each control fragment, i.e., four for $\mathcal{C}_1$, three for $\mathcal{C}_2$, and so forth. These are then used to develop a linear (affine in practice) regression model for each control fragment that predicts $\boldsymbol{a}$ as a linear function of $\boldsymbol{s}$. This resulting predictive model then becomes the control policy, $\boldsymbol{\pi}$, for the control fragment. This control policy is then used to help inform the next round of motion reconstruction using Guided SAMCON.

In the following section, we describe how the iterative use of the linear regression model can be understood as being an EM-based (expectation maximization) policy search algorithm. Alternatively, readers can choose to jump directly to the specific details of the linear regression for our problem, as described in §5.2.

5.1 Guided Learning as EM-based Policy Search

Starting from a state $\boldsymbol{s}_{k-1}$, each execution of a control fragment $\mathcal{C}_k$ results in a simulation tuple $\tau = (\boldsymbol{s}_{k-1}, \boldsymbol{a}_k, \boldsymbol{s}_k)$. Given a reward function $R(\tau)$ that measures the goodness of this execution, policy search seeks for the optimal policy that maximizes the expected return

$$J(\boldsymbol{\theta}) = \int_\tau P(\tau;\boldsymbol{\theta})R(\tau) \tag{3}$$

with respect to the feedback parameters $\boldsymbol{\theta}$. The probability density of a simulation tuple is determined by:

$$P(\tau;\boldsymbol{\theta}) = P(\boldsymbol{s}_k|\boldsymbol{s}_{k-1}, \boldsymbol{a}_k)\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})P(\boldsymbol{s}_{k-1}) \tag{4}$$

where $P(\boldsymbol{s}_k|\boldsymbol{s}_{k-1}, \boldsymbol{a}_k)$ is the transition probability density and $\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})$ represents the probability density of the feedback action given the start state and the feedback parameters. We model $\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})$ as Gaussian explorations superimposed onto the deterministic feedback policy of Equation 2, i.e.:

$$\begin{aligned}\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}) &:= \boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)\\ &\sim \mathcal{N}(\boldsymbol{M}_k\boldsymbol{s}_{k-1} + \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)\end{aligned} \tag{5}$$

The feedback parameters are then defined as $\boldsymbol{\theta} = \{\boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k\}$. We use a diagonal covariance matrix $\boldsymbol{\Sigma}_k$ with the assumption that each dimension of the action space is independent.

An EM-style algorithm offers a simple way to find the optimal policy by iteratively improving the estimated lower-bound of the policy's expected return. [Peters and Schaal 2007] applies an EM algorithm to episodic policy search for a linear policy and shows that the iterative update procedure is just a weighted linear regression over the execution episodes of the current policy. Specifically, let $\boldsymbol{\theta}_0$ be the current estimation of the policy parameters, EM-based policy search computes a new estimation $\boldsymbol{\theta}$ that maximizes:

$$\log\frac{J(\boldsymbol{\theta})}{J(\boldsymbol{\theta}_0)} = \log\int_\tau P(\tau;\boldsymbol{\theta})R(\tau)/J(\boldsymbol{\theta}_0) \tag{6}$$

$$\geq \frac{1}{J(\boldsymbol{\theta}_0)}\int_\tau P(\tau;\boldsymbol{\theta}_0)R(\tau)\log\frac{P(\tau;\boldsymbol{\theta})}{P(\tau;\boldsymbol{\theta}_0)} \tag{7}$$

$$\propto \int_\tau P(\tau;\boldsymbol{\theta}_0)R(\tau)\log\frac{\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})}{\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}_0)} \tag{8}$$

$$= L(\boldsymbol{\theta};\boldsymbol{\theta}_0) + C \tag{9}$$

where Equation 7 applies Jensen's inequality to the concave logarithm function, C is a constant independent of $\boldsymbol{\theta}$ and

$$L(\boldsymbol{\theta};\boldsymbol{\theta}_0) := \int_\tau P(\tau;\boldsymbol{\theta}_0)R(\tau)\log\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}) \tag{10}$$

Note that the optimal $\boldsymbol{\theta}$ must satisfies $J(\boldsymbol{\theta}) \geq J(\boldsymbol{\theta}_0)$ because Equation 7 is always zero when $\boldsymbol{\theta} = \boldsymbol{\theta}_0$. $L(\boldsymbol{\theta};\boldsymbol{\theta}_0)$ can be further estimated from a number of simulation tuples $\{\tau_k^i\}$ sampled according to the current policy $\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}, \boldsymbol{\theta}_0)$ as:

$$L(\boldsymbol{\theta};\boldsymbol{\theta}_0) \approx \frac{1}{N_k}\sum_{i=1}^{N_k} R(\tau^i)\log\boldsymbol{\pi}_k(\boldsymbol{a}_k^i|\boldsymbol{s}_{k-1}^i;\boldsymbol{\theta}) \tag{11}$$

where N_k is the number of such tuples. By letting $\partial L(\boldsymbol{\theta};\boldsymbol{\theta}_0)/\partial\boldsymbol{\theta} = \mathbf{0}$ we can find the locally optimal estimation of $\boldsymbol{\theta}$ by solving

$$\begin{aligned}\mathbf{0} &= \frac{\partial}{\partial\boldsymbol{\theta}} L(\boldsymbol{\theta};\boldsymbol{\theta}_0)\\ &\propto \sum_{i=1}^{N_k} R(\tau^i)\frac{\partial}{\partial\boldsymbol{\theta}}\log\boldsymbol{\pi}_k(\boldsymbol{a}_k^i|\boldsymbol{s}_{k-1}^i;\boldsymbol{\theta}) \qquad (12)\end{aligned}$$

With this maximization step in place (the M step), we then update $\boldsymbol{\theta}_0$ with this optimal $\boldsymbol{\theta}$ and then recompute a new set of samples (the E step) and repeat the EM iteration until obtaining optimal policies.

As we are learning the feedback policies against the random walk $\mathcal{W}$, the sample tuples $\{\tau_k^i\}$ for all the control fragments $\{\mathcal{C}_k\}$ can be collected simultaneously by generating a long successful execution of $\mathcal{W}$, represented by $\boldsymbol{\tau} = \{\tau_{k_1}, \tau_{k_2}, \dots\}$, and then extracting simulation tuples for each individual control fragment from it. Figure 5 provides a simple sketch of this procedure. Furthermore, we assign a constant reward to all such tuples, which implies a special reward function in the form of

$$R(\tau) = \begin{cases} 1 & \text{tuple } \tau \text{ is good enough in the long run so} \\ & \text{that the random walk } \mathcal{W} \text{ can succeed.} \\ 0 & \text{otherwise.}\end{cases} \qquad (13)$$

Solving Equation 12 against this reward function and the Gaussian Explorations of Equation 5 leads to the linear regression that we describe next.

5.2 Estimation of Linear Feedback Policy

The linear regression problem solved for control fragment k yields a model to predict $\boldsymbol{a}$ as an affine function of $\boldsymbol{s}$, as per equation 2, where

$$\boldsymbol{M}_k = \left[(S_k^T S_k)^{-1}(S_k^T A_k)\right]^T \qquad (14)$$

$$\hat{\boldsymbol{a}}_k = \bar{\boldsymbol{a}}_k - \boldsymbol{M}_k\bar{\boldsymbol{s}}_{k-1} \qquad (15)$$

$$\operatorname{diag}(\boldsymbol{\Sigma}_k) = \frac{1}{N_\tau}\operatorname{diag}\left[(A_k - S_k\boldsymbol{M}_k^T)^T(A_k - S_k\boldsymbol{M}_k^T)\right] \qquad (16)$$

where $\bar{\boldsymbol{a}}_k$ and $\bar{\boldsymbol{s}}_{k-1}$ are the averages of $\boldsymbol{a}_k^i$ and $\boldsymbol{s}_{k-1}^i$ respectively, the N_k-row matrices S_k and A_k represent the centered collections of all the tuples, i.e.

$$S_k = \left[\boldsymbol{s}_{k-1}^1 - \bar{\boldsymbol{s}}_{k-1}, \dots, \boldsymbol{s}_{k-1}^{N_k} - \bar{\boldsymbol{s}}_{k-1}\right]^T \qquad (17)$$

$$A_k = \left[\boldsymbol{a}_k^1 - \bar{\boldsymbol{a}}_k, \dots, \boldsymbol{a}_k^{N_k} - \bar{\boldsymbol{a}}_k\right]^T \qquad (18)$$

To prevent the regression from being underdetermined, the random walk is generated to be long enough so that $N_k \geq 200$ for all control fragments. We further regularize the Frobenius norm of the feedback gain matrix $\boldsymbol{M}_k$, so that

$$\boldsymbol{M}_k = \left[(S_k^T S_k + \lambda I)^{-1}(S_k^T A_k)\right]^T \qquad (19)$$

is used instead of Equation 14. The regularization coefficient $\lambda = 10^{-6}$ in all our experiments.

We further use the observed prediction variances, as captured by $\boldsymbol{\Sigma_k}$, to provide a stochastic version of the control policy (cf. §5.1) that will be used to guide the sampling used by the search algorithm (SAMCON) that serves as our control oracle:

$$\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}) := \mathcal{N}(\boldsymbol{M}_k\boldsymbol{s}_{k-1} + \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)$$

Algorithm 1 Guided SAMCON

Input:
1: a random walk on the control graph $\mathcal{W} = \{\mathcal{C}_k\}, k = 1, \dots, N$
2: the start state $\boldsymbol{s}_0$
Output: a successful execution of the sequence $\boldsymbol{\tau}$

```
1: $\{\boldsymbol{s}_0^j\} \leftarrow$ initialize the starting set with $N_s$ replicas of $\boldsymbol{s}_0$
2: for $k \leftarrow 1$ to $N$ do
3:     for each sample $j$ do
4:         generate action $\boldsymbol{a}_k^j \sim \boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}^j) \sim \mathcal{N}(\boldsymbol{M}_k\boldsymbol{s}_{k-1}^j + \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)$
5:         $\boldsymbol{s}_k^j \leftarrow$ execute control fragment $\mathcal{C}_k$ against $\boldsymbol{a}_k^j$
6:         record a simulation tuple $\tau_k^j = (\boldsymbol{s}_{k-1}^j, \boldsymbol{a}_k^j, \boldsymbol{s}_k^j)$
7:         $E_k^j \leftarrow$ evaluate end state $\boldsymbol{s}_k^j$
8:     end for
9:     $\{\tau_k^{j*}\} \leftarrow$ select $n_s$ elite samples according to $\{E_k^j\}$
10:    $\{\boldsymbol{s}_k^j\} \leftarrow$ resample $\{\boldsymbol{s}_k^{j*}\}$ to get a new starting set of size $N_s$
11: end for
12: $\boldsymbol{\tau} = \{\tau_k\} \leftarrow$ select the best path from all saved $\{\tau_k^{j*}\}$
```

5.3 Guided SAMCON

A key to the success of the guided learning is that it learns from *useful* examples, i.e., only those that result from successful completions of the random walk in every iteration. Levine and Koltun [2013; 2014] suggest that trajectory optimization can be used to collect such useful examples for guided policy search. We draw inspiration from this type of guided learning, to develop a form of guided learning that relies on sampling-based motion control (SAMCON) methods, as proposed by [Liu et al. 2010; Liu et al. 2015] to provide successful executions of the target random walk. SAMCON allows us to work with long sequences of complex motions, and has proved to be capable of generating the controls for a wide variety of motion skills. We call this new method *Guided SAMCON*.

We first begin by reviewing SAMCON. For this review, the exact index of the control fragment is unimportant and thus we represent the random walk according to their sequence index in the random walk, i.e., $\mathcal{W} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_N\}$, where $N = \sum_{k=1}^K N_k$ is the length of the random walk, and N_k is the number of times a given control fragment k appears in the random walk. Beginning from the initial state of the first control fragment, $\mathcal{C}_1$, we utilize SAMCON to develop a sequence of actions $\{\boldsymbol{a}_k\}$ that results in control fragment end states, $\{\boldsymbol{s}_k\}$, that are nearby those of the desired reference motions. The set of simulation tuples $\{(\boldsymbol{s}_{k-1}, \boldsymbol{a}_k, \boldsymbol{s}_k)\}$ then describes the samples from the same control fragment k that are collected together for regression. Note that this represents a slight abuse of notation in that we use $\boldsymbol{s}_{k-1}$ to refer to the previous state in the random walk sequence, rather than a true control fragment index, whose identification numbering can be in an arbitrary order. However, this significantly simplifies the notation for the remainder of the paper.

Algorithm 1 outlines the main steps of SAMCON, and Figure 5 provides a simple example. SAMCON can be viewed as a type of Sequential Monte Carlo algorithm [Doucet and Johansen 2011]. Specifically, for the first control fragment, SAMCON initializes an initial set of states $\{\boldsymbol{s}_0^j\}$ with $j \in 1...N_s$ replicas of the start state $\boldsymbol{s}_0$, and samples an action $\boldsymbol{a}_1^j$ for each $\boldsymbol{s}_0^j$ according to a sample distribution $\boldsymbol{\pi}(\boldsymbol{a}_1|\boldsymbol{s}_0)$. It then advances the simulation from $\boldsymbol{s}_0^j$ while executing the control fragment with the corresponding compensation offset $\Delta\hat{\boldsymbol{p}}_1^j$ computed from $\boldsymbol{a}_1^j$ as described in Section 4. The simulation results in a tuple $\tau_1^j = (\boldsymbol{s}_0^j, \boldsymbol{a}_1^j, \boldsymbol{s}_1^j)$ whose end state $\boldsymbol{s}_1^j$ is evaluated according to its similarity to the reference end state that corresponds to the control fragment.

We measure this similarity by a cumulative cost function:

$$E = w_p E_p + w_r E_r + w_e E_e + w_b E_b + w_c E_c + w_v E_v + w_L E_L + w_a E_a \quad (20)$$

where the terms for pose control E_p, root control E_r, end-effector control E_e, and balanced control E_b are identical to the original work [Liu et al. 2010]. We additionally regularize the differences between the simulation and the reference in terms of centroid position E_c, centroid velocity E_v, and the angular momentum E_L. The last term, E_a, simply serves to regularize the Euclidean norm of the actions. We use $(w_p, w_r, w_e, w_b, w_c, w_v, w_L, w_a) = (4.0, 4.0, 10.0, 1.0, 3.0, 0.1, 0.03, 0.5)$ for all our experiments. Our results are not sensitive to the exact values of these weights so other values within the same order of magnitude may be used as well.

After executing all sample actions and obtaining N_s simulation tuples $\{\tau_1^j\}$, guided SAMCON selects and saves the n_s best tuples $\{\tau_1^{j*}\}$, as measured by the lowest cumulative costs, and then systematically resamples the corresponding end states $\{\boldsymbol{s}_1^{j*}\}$ according to their costs to obtain a new starting set $\{\boldsymbol{s}_1^j\}$ of size N_s for the successive control fragment. This is akin to the resampling procedure used in particle filtering, i.e., better samples produce more successors. This sampling procedure is repeated for each stage of the motion, i.e., once per control fragment, until the end of the random walk is reached. Finally, the resultant execution episode $\boldsymbol{\tau} = \{\tau_k\}$ is chosen to be the best path of all saved tuples $\{\tau_k^{j*}\}$.

Guided SAMCON uses the current policy of every control fragment $\mathcal{C}_k$ as the distribution to sample from, i.e., $\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) = \boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}; \boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k) \sim \mathcal{N}(\boldsymbol{M}_k\boldsymbol{s}_{k-1} + \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)$. This can be viewed as an enhancement of the original SAMCON algorithm [Liu et al. 2010] that employed a fixed sampling distribution, $\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$, and also of the improved SAMCON algorithm [Liu et al. 2015] that evolves the mean and covariance of the sample distributions iteratively in a state-independent fashion, i.e., $\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) \sim \mathcal{N}(\hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k)$. The guided sample selection and resampling implicitly focuses the exploration on regions of the state space that are both relevant to the current policy as well as regions of the action space that are known to yield desired motions.

Voluntarily including noise in optimization has been shown to be useful to prevent over-fitting and allows the learned policy to deal with larger uncertainty [Wang et al. 2010; Liu et al. 2012]. We build on this idea by further adding a Gaussian noise vector $\boldsymbol{\varepsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma_\varepsilon^2 I)$ to the action samples. We thus compute the compensation offset $\Delta\hat{\boldsymbol{p}}_k^j$ from $\boldsymbol{a}_k^j + \boldsymbol{\varepsilon}_k$. The noise vector is assumed to be unknown to the feedback policies, and is not recorded or included in regression. We find that a uniform setting $\sigma_\varepsilon = 3°$ is enough to allow all of our motions to be robustly executed.

5.4 Learning Control Graphs

Algorithm 2 summarizes the whole guided learning framework of control graphs. Given several example motion clips of the target skills as input, the pipeline builds a control graph that synthesizes robust dynamic motions from arbitrary random walks over the graph. This allows for motion planners, which are beyond the scope of this paper, to work with the graph as a simple high-level abstraction of the motion capabilities. The whole pipeline consists of the following sub-procedures:

Building control graphs: A reference motion graph is firstly built (line 1 of Algorithm 2), and then converted to a control graph (line 2) as described in Section 4. Building high-quality motion graphs can be a non-trivial task, even with the help of automated techniques such as the one proposed by [Kovar et al. 2002]. Manual

Algorithm 2 Guided Learning Pipeline

Input: example motion clips of skills
Output: a control graph $\mathcal{G}$

```
 1: build a reference motion graph $\tilde{\mathcal{G}}$ from input motion clips
 2: initialize a control graph $\mathcal{G} = \{\mathcal{C}_k\}$ according to $\tilde{\mathcal{G}}$
 3: generate a random walk $\mathcal{W} = \{\mathcal{C}_{k_1}, \dots, \mathcal{C}_{k_N}\}$
 4: refine the open-loop control clip $\hat{m}_k$ for every $\mathcal{C}_k$
 5: initialize $\boldsymbol{M}_k = \mathbf{0}$, $\boldsymbol{a}_k = \mathbf{0}$, $\boldsymbol{\Sigma}_k = \sigma_0^2 I$ for every $\mathcal{C}_k$
 6: for every EM iteration do                                   ▷ policy search
 7:     generate a successful execution $\boldsymbol{\tau}$ of $\mathcal{W}$ with Guided SAMCON
 8:     for each control fragment $\mathcal{C}_k$ do
 9:         $\{\tau_k^i\}$ ← extract sample simulation tuples of $\mathcal{C}_k$ from $\boldsymbol{\tau}$
10:         update $\boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k$ by linear regression on $\{\tau_k^i\}$
11:     end for
12: end for
```

tuning is often necessary to achieve natural-looking transitions and to remove artifacts such as foot-skating. Fortunately, the usage of simulation naturally offers the ability to produce physically plausible motions for the control graph. Therefore, the reference motion graph does not necessarily need to be carefully tuned. In this paper, we simply specify the connectivity of the motion graphs for our control graphs manually. We kinematically blend a few frames of the relevant motion clips near the transition points. Our learning procedure is robust to kinematic flaws due to blending or noise, and is able to generate high-quality simulated motions.

Refining open-loop control clips: The initial control clips of every control fragment are directly computed from the individual motion capture example clips, which are not necessary physically plausible for the transitions in the control graph. To facilitate the graph learning process, we further refine these open-loop controls as indicated on line 4. Specifically, this is done by performing the original SAMCON on the motion sequence corresponding to the random walk $\mathcal{W}$, and then replacing the initial open-loop control clip $\hat{\boldsymbol{m}}_k$ and the reference end states with the average of all simulation instances of the control fragment $\mathcal{C}_k$ in $\mathcal{W}$. The averaging not only reduces the noise due to random sampling as suggested by [Liu et al. 2015], but also maximizes the possibility of finding a robust feedback policy that can deal with all possible transitions.

Learning Feedback Policies: In line 5, the feedback policies are initialized, as well as the default exploration covariances. We find that $\sigma_0 = 5°$ works for all the skills that we have tested. The EM-based policy search is performed in line 6–12, where the guided SAMCON trials and the linear regressions are alternated to improve the feedback policies iteratively. In all our experiments, this policy search process can converge to robust feedback policies in at most 20 iterations. Guided SAMCON can occasionally fail when generating a long sequence of random walk, especially in the first iteration where the initial policy is applied. To mitigate this problem, we generate more samples ($N_s = 1000$) per stage during the first iteration than for the successive iterations ($N_s = 200$). If the algorithm fails to complete the designated graph walk, we roll back the execution of the latest three controllers (25-50 control fragments) and then restart guided SAMCON from that point.

6. RESULTS

We have implemented our framework in C++. We augment the Open Dynamic Engine (ODE) v0.12 with an implicit damping scheme [Liu et al. 2013] to simulate the character faster and more stably. On a desktop with an Intel Core i5 @ $2.67GHz$ CPU, our single threaded implementation runs at $10\times$ real-time using a simulation time step of $5ms$. Except for the retargeting experiments, all our experiments are performed with a human model that is $1.7m$

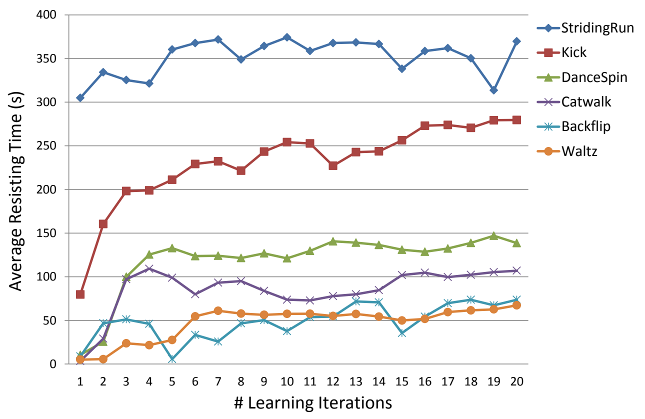


Fig. 6: The relationship between the number of learning iterations and the controller robustness as indicated by the resisting duration before failure under random pushes of growing magnitude.

Skills	PD-Gains Set	T_{cycle} (s)	t_{learning} (min)	t_{f} (s)
Catwalk	(a)	0.7	40.3	107
StridingRun	(a)	0.45	21.1	370
Waltz	(a)	5.0	314	67.2
Kick	(b)	1.6	93.8	280
DanceSpin	(b)	1.6	102	139
Backflip	(b)	2.5	153	73.7

Table II. : Performance statistics for cyclic motions. T_{cycle} represents the length of a reference cycle. t_{learning} is the learning time for each skill on a 20-core computer. t_{f} represents the average resisting duration before failure under random pushes of growing magnitude.

tall and weights $62kg$. It has 45 DoFs in total, including 6 DoFs for the position and orientation of the root. Two sets of PD-gains are used in our experiments: (a) for basic locomotion, we simply set $k_p = 500$, $k_d = 50$ for all joints; (b) for highly dynamic stunts, a stronger waist ($k_p = 2000, k_d = 100$) and leg joints ($k_p = 1000, k_d = 50$) are necessary.

6.1 Cyclic Skills

The simplest non-trivial control graphs are those built from individual cyclic skills. A variety of cyclic skills have been tested to fully evaluate the capability of the proposed learning framework, including basic locomotion gaits, dancing elements, flips, and kicks. The example motion clips for these skills are from various sources and were captured from different subjects. We simply apply them onto our human model, and kinematically blend the beginning and end of the clips to obtain cyclic reference motions. Errors due to model mismatches and blending are automatically handled by our physics-based framework. The animation sequences shown in Figure 7 demonstrate the executions of several learned skills. We encourage readers to watch the supplemental video to better evaluate the motion quality and robustness of the controllers.

The offline learning is performed on compute clusters with tens of cores. The performance of the learning pipeline is determined by the number of necessary runs of guided SAMCON, whose computational cost scales linearly with respect to the length of the clip, the number of samples N_s, and inversely with the number of cores available. Table II lists the learning time for several cyclic skills, measured on a small cluster of 20 cores. Note that here we run the learning pipeline with the same configuration for all motions for ease of comparison, i.e., 20 iterations of guided learning, with $N_s = 1000$ in the first iteration and $N_s = 200$ for the remaining iterations. In practice, the required SAMCON samples can be much lower, e.g. to $N_s = 50 \sim 100$, after the first few learning iterations, as the feedback policies usually converge quickly under the guided learning.

The learned skills are robust enough to enable repeated executions even under external perturbations. In the supplemental video we show that $400N \times 0.2s$ impulses can be applied to the character's trunk during the flight phase of kicking without causing the motion to fail. To more systematically test the robustness of the learned controllers, we apply a sequence of horizontal pushes in random directions to the character's trunk, and measure the time that the motion skill can last before the character falls. The magnitude of the perturbation force is generated from a normal distribution with increasing variance. This experiment is performed 100 times, and the average performance is computed as an indication of robustness as shown in the last column of Table II. Figure 6 further illustrates how the robustness improves as a function of the number of guided policy iterations. Generally speaking, faster motions such as running and kicking take less learning iterations to achieve stable cyclic skills that can execute indefinitely, as well as tolerate larger perturbations. In contrast, slow motions such as the catwalk and the waltz are more sensitive to perturbations.

All the test skills can be learned with the standard settings as described in the previous sections, while special treatment is applied for walking and running in order to achieve symmetric gaits. Specifically, we pick one stride (half step) from the example clip and concatenate its mirror stride to generate a symmetric reference motion. In addition, we only learn the feedback policies for the first stride, and mirror the states and actions for the second stride so that the feedback policies are symmetric too. We further employ contact-aligned phase-resetting for walking and running controllers, which improves the robustness to large perturbations. Interestingly, we found the contact-aligned phase-resets not helpful for learning controllers for complex skills such as kicks and backflips, which may indicate that the contact events are not informative phase indicators for such motions. Another interesting observation of the learned walking and running controllers is that the character turns when gentle sideways pushes are applied. This offers a simple way to parameterize these locomotion skills, as we can record the corresponding actions under external pushes and add them voluntarily to the action vectors in order to make the character turn in moderate speed. We use this simple parameterization method to achieve basic steering behaviors in our demos. For rapid turns we still need to use controllers learned from relevant motion capture examples.

The robustness of the learning framework enables additional interesting applications. For example, in the supplemental video we show that two significantly different backflips can be learned from a single motion capture example, where one is learned from a shorter reference cycle than the other. The guided learning process automatically finds a physically feasible movement that fills in the missing segment of the shorter reference trajectory. Our framework also supports retargeting controllers onto characters with significantly different morphology from the motion captured subjects. We simply re-run the pipeline on the new character, with the *open-loop clip refinement* step warm-started from the results built for our default character. Figure 8 shows several examples where we retarget the cyclic kick and the dance spin to characters with modified body

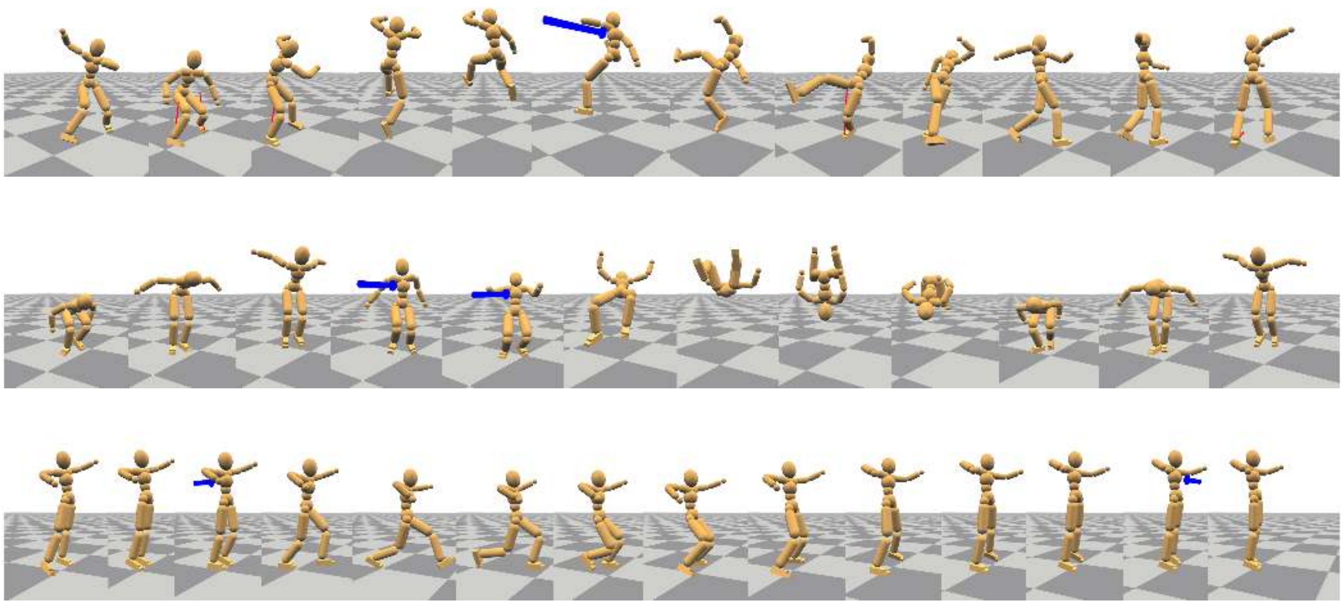


Fig. 7: Simulations of learned skills under external pushes. Top: kick. Middle: backflip. Bottom: waltz.

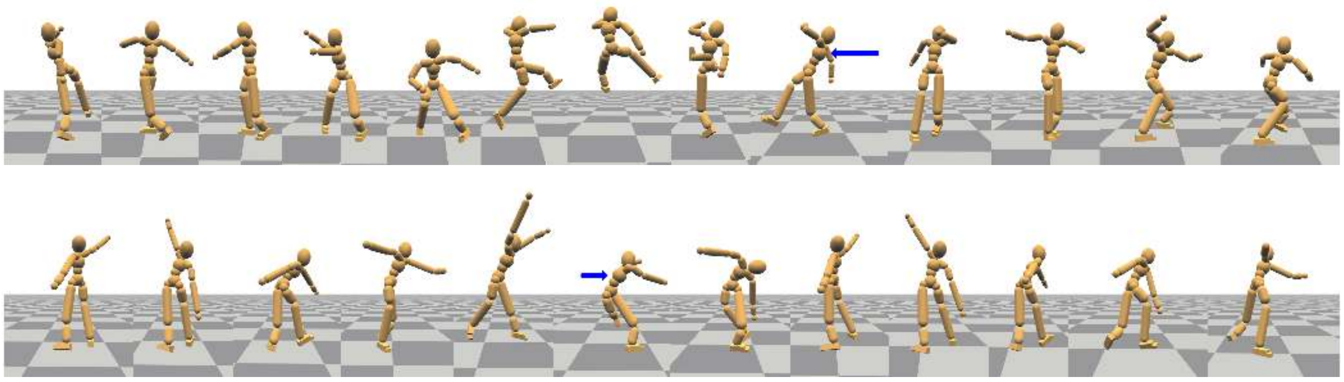


Fig. 8: Retargeting kick (top) and dance-spin (bottom) to characters with modified body ratio.

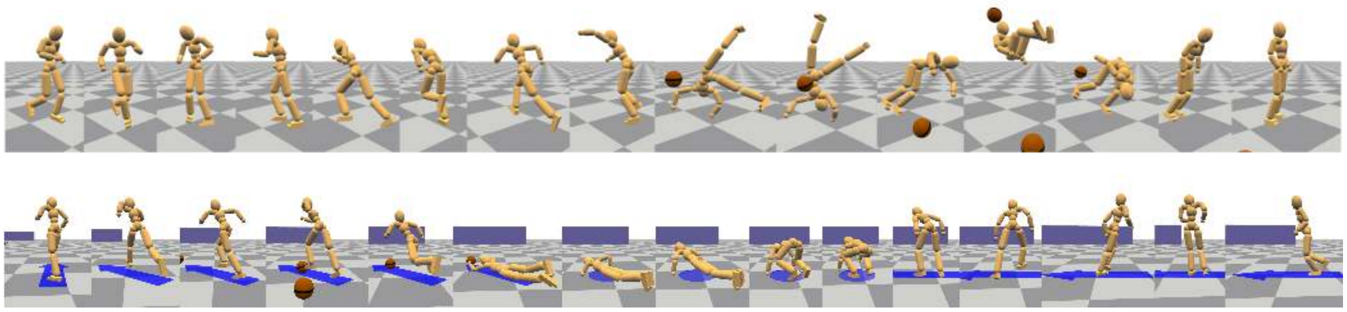


Fig. 9: Applications of the control graph. Top: random walk with external perturbations. Bottom: steering.

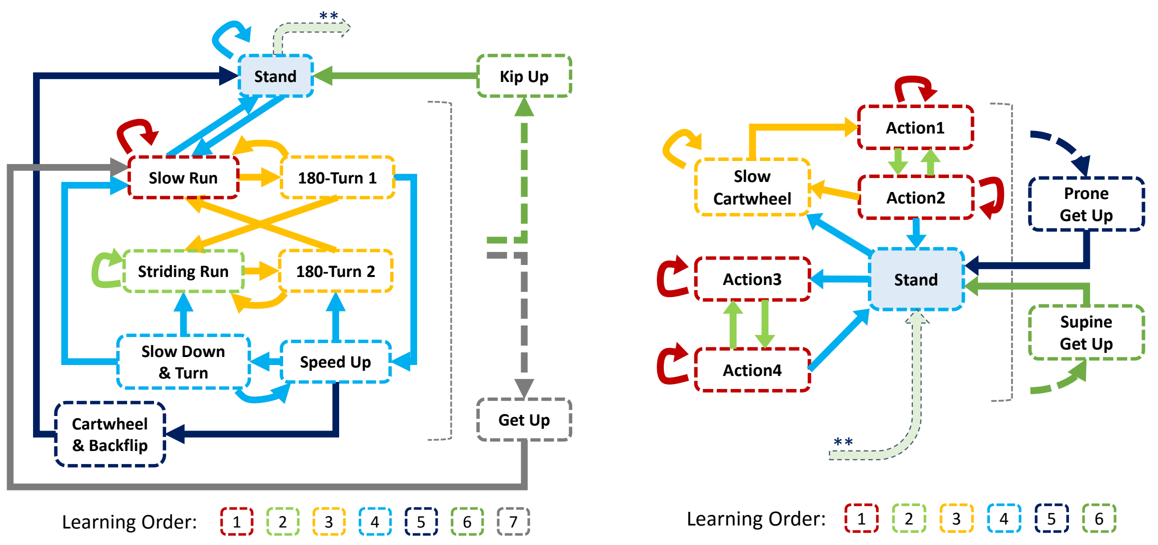


Fig. 10: Two porototype control graphs progressively learned in the order marked in different colors. Only major controllers are shown in the graph for clarity. The rising skills pointed by dashed arrows will be triggered automatically once the character falls. Left: Locomotion and gymnastic graph. Right: Bollywood dancing graph. Action1–arm hip shake; Action2–chest pump+swag throw; Action3–pick and throw; Action4–hand circular pump.

segment lengths. The retargeted controllers are robust to external perturbations as before.

6.2 Control Graphs

Figure 10 shows two prototype control graphs, one consisting of runs, turns, gymnastic movements, and balancing, and the other consisting of Bollywood dancing elements, get-up motions, and balancing. Only major controllers are shown in the graphs for clarity. The two control graphs can be further composed into a larger one through the standing behavior. Learning the controllers for the entire control graphs all at once can be inefficient because different controllers converge in different speed, i.e., some controllers quickly become robust, while others may cause SAMCON to fail and restart constantly. This disparity results in excessive samples being used for the easy controllers and excessive restarts for the difficult ones, if the entire control graph were to be learned all at once. To mitigate this problem, we learn the control graph progressively. Figure 10 illustrates the example learning orders we use. Specifically, we start from learning controllers for a few cyclic skills, using the process described in the last subsection. Non-cyclic skills are then gradually incorporated into the latest subgraph by rerunning the whole learning pipeline. This progressive process skips the learned skills from guided SAMCON by merely executing the learned policies instead of generating additional exploratory samples for their learning. Another scheme we employ to improve learning efficiency is to generate random walks that visit each skill with approximately equal probability. Some connections between the learned skills are temporarily neglected to achieve this condition. Our experiments show that both control graphs can be learned within one night in the fashion described above.

We further include a few rising skills in the control graphs that will be automatically executed when the character falls. These rising skills have only one-way connections with the graph and we learn them in a separate procedure. We create learning cycles by pushing the character on the trunk in suitable directions and then invoke a ragdoll controller that tracks the starting pose of the target rising skill. When the difference between the simulated pose and this starting pose is small enough, the rising skill is activated and the character gets up and once again transitions to the beginning of the learning cycle. We currently use a simple *fall detection* algorithm that monitors the magnitude of the action vector as computed by the feedback policies. Once this exceeds a fixed threshold, we activate the ragdoll control followed by an appropriate rising skill.

We show several applications of the prototype control graphs in the supplemental video. The learned skills in the graph are quite robust: a random walk on the graph can always succeed when no perturbations are applied. With the help of a simple greedy planner, we can easily achieve interactive navigation in the scene. The characters can also robustly perform the desired motions in the presence of moderate external perturbations such as pushes on the trunk and ball-impacts as shown in Figure 9. The character will fall if it is disturbed too much, which automatically activates the rising controllers that return the character to performing desired motions designated by the high-level planner.

Figure 11 demonstrates two simulated characters, steered by a high-level planner, to always try to run into each other. They repeat the overall behaviors of colliding, falling, and getting up. The complex contacts and interactions between the characters would be too difficult to synthesize via kinematic approaches, while our framework can easily generate these motions in real-time thanks to the physics-based nature of the simulations and the robustness of the control graphs. In the video, we further show another example involving four characters, all simultaneously simulated in real-time, that perform the same overall interaction behaviors.

7. DISCUSSION

We have introduced a general framework that learns and organizes physics-based motion skills from example motion capture clips. Key to achieving the results is the use of control graphs composed of control fragments, the use of random walks on the control graphs

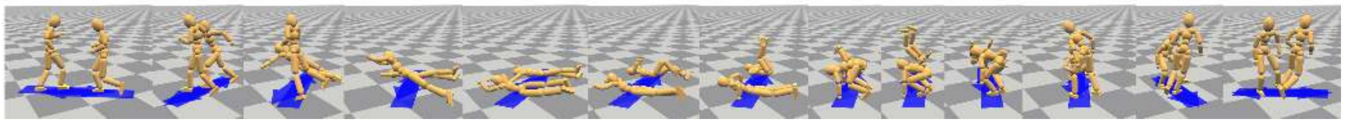


Fig. 11: Two simulated characters try to run into each other. Both of them are controlled by the same control graph.

for learning, and the use of guided policy search for developing linear feedback policies for the control fragments. To the best of our knowledge, this is the first time that a very diverse range of motions, including locomotion, highly dynamic kicks and gymnastics, standing, and rising motions, can be synthesized in real time for 3D physics-based characters and controlled and integrated together in a uniform framework. We believe the proposed method can be easily applied to a variety of other skilled motions. This offers a potential solution to the difficulties in developing general control methods that still prevent the more widespread adoption of physics-based methods for character animation.

A primary finding of our work is that sequences of linear feedback policies based on a fixed set of state features and action features, as implemented by the control fragments, do well at controlling many skills, including not only the basic locomotion, but also rising skills and complex highly-agile movements. It is further interesting to note that these linear policies can be learned from a suitable data stream using standard linear regression methods. Two components of the success are: (a) the ability to generate high-quality open-loop motion reconstructions using SAMCON; and (b) the use of guided learning which effectively selects samples in the vicinity of the states and actions produced by the policy and therefore encourages convergence between the developed offline solutions and the learned linear policies.

In practice, every run of the learning pipeline on a given skill usually results in different policies, which suggests that the feedback policies may have a low-rank structure that admits multiple solutions. This has been proven to be true for basic locomotion [Ding et al. 2015; Liu et al. 2012]. Learning phase-specific reduced-order policies for complex skills is an interesting topic for future work.

Our current state features and action features were selected with skills in mind such as locomotion, kicks, and dancing, these all being skills where the character's legs are extensively used for balance. However, these features proved to be suitable for a wider range of skills, including those where the arms play an important role, e.g., cartwheels and rising up motions. For motions that are dominated by the control applied to the arms, such as a hand-stand or a hand-stand walk, we expect that some new state features and action features may need to be introduced.

A sequence of control fragments, or a controller, implicitly defines an time-indexed piece-wise linear feedback policy, with an approximate $0.1s$ time interval, as inherited from the original SAMCON algorithm [Liu et al. 2010]. The feedback is therefore dependent on both the current simulation state and the time index in the reference motions. This scheme mitigates many difficulties in learning the feedback policies, but makes the learned policies less flexible. As future work, we wish to develop state-based feedback policies from executions of the time-indexed policies by leveraging more complex policy representations, such as neural networks [Levine and Koltun 2013; Mordatch and Todorov 2014; Tan et al. 2014].

We wish to develop and integrate parameterized versions of the motions and their feedback controllers. Parameterization is also another form of generalization, and an appropriate learning process can likely be bootstrapped from the initial unparameterized motions. Currently, our ability to steer the character is developed in an ad hoc fashion. Parameterization with continuous optimization [Yin et al. 2008; Liu et al. 2012] and interpolations between controllers [da Silva et al. 2009; Muico et al. 2011] may help enrich the variance of learned skills. It may also be possible to integrate the use of abstract models, such as the inverted pendulum model [Coros et al. 2010] or feature-based control [Mordatch et al. 2010] in support of generalization with respect to larger perturbations or motion parameterization.

The efficiency of our current learning pipeline could potentially be improved in several respects as many sample simulations are discarded without being fully exploited. For example, guided SAMCON discards all the simulation tuples except those belong to the best path, and even the saved simulation tuples are discarded after their use in the linear regression for the current iteration of the guided learning. These samples could likely be further utilized to reduce the necessary duration of random walk and to enhance the robustness of the learned policies as they offer extra information about the policy space. Reusing these samples with importance weights [Hachiya et al. 2009] offers one possible path forward to developing a more efficient learning process.

After the initialization procedures, the current framework is largely automated, with uniform parameter settings being used to develop most of the motions. However, manually designing the reference motion graph is still necessary at the beginning of the pipeline. Developing good open-loop control clips for difficult skills or from poor-quality reference motions remains the part of the learning pipeline that still requires some manual intervention. For future work, we would like to create a fully automated pipeline.

ACKNOWLEDGMENTS

This project is partially supported by NSERC Discovery Grants Program RGPIN-2015-04843 and Singapore Ministry of Education Academic Research Fund, Tier 2 (MOE2011-T2-2-152).

REFERENCES

AL BORNO, M., DE LASA, M., AND HERTZMANN, A. 2013. Trajectory optimization for full-body movements with complex contacts. *TVCG 19,* 8, 1405–1414.

AL BORNO, M., FIUME, E., HERTZMANN, A., AND DE LASA, M. 2014. Feedback control for rotational movements in feature space. *Computer Graphics Forum 33,* 2.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2009. Robust task-based control policies for physics-based characters. *ACM Trans. Graph. 28,* 5 (Dec.), 170:1–170:9.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2010. Generalized biped walking control. *ACM Trans. Graph. 29,* 4 (July), 130:1–130:9.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Simulation of human motion data using short-horizon model-predictive control. In *Computer Graphics Forum*. Vol. 27. Wiley Online Library, 371–380.

DA SILVA, M., DURAND, F., AND POPOVIĆ, J. 2009. Linear bellman combination for control of character animation. *ACM Trans. Graph. 28,* 3 (July), 82:1–82:10.

DE LASA, M., MORDATCH, I., AND HERTZMANN, A. 2010. Feature-based locomotion controllers. *ACM Trans. Graph. 29,* 4 (July), 131:1–131:10.

DING, K., LIU, L., VAN DE PANNE, M., AND YIN, K. 2015. Learning reduced-order feedback policies for motion skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '15. ACM, New York, NY, USA, 83–92.

DOUCET, A. AND JOHANSEN, A. M. 2011. A tutorial on particle filtering and smoothing: Fifteen years later. In *Handbook of Nonlinear Filtering*. Oxford, UK: Oxford University Press.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001*. 251–260.

GEIJTENBEEK, T. AND PRONOST, N. 2012. Interactive character animation using simulated physics: A state-of-the-art review. In *Computer Graphics Forum*. Vol. 31. Wiley Online Library, 2492–2515.

GEIJTENBEEK, T., VAN DE PANNE, M., AND VAN DER STAPPEN, A. F. 2013. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG) 32,* 6, 206.

HA, S., YE, Y., AND LIU, C. K. 2012. Falling and landing motion control for character animation. *ACM Trans. Graph. 31,* 6 (Nov.), 155:1–155:9.

HACHIYA, H., PETERS, J., AND SUGIYAMA, M. 2009. Efficient sample reuse in EM-based policy search. In *Machine Learning and Knowledge Discovery in Databases*. Lecture Notes in Computer Science, vol. 5781. Springer Berlin Heidelberg, 469–484.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of SIGGRAPH*. ACM, New York, NY, USA, 71–78.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, USA, 473–482.

KWON, T. AND HODGINS, J. 2010. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *SCA*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 129–138.

LEE, J. AND LEE, K. H. 2006. Precomputing avatar behavior from human motion data. *Graphical Models 68,* 2, 158–174.

LEE, Y., KIM, S., AND LEE, J. 2010. Data-driven biped control. *ACM Trans. Graph. 29,* 4 (July), 129:1–129:8.

LEE, Y., WAMPLER, K., BERNSTEIN, G., POPOVIĆ, J., AND POPOVIĆ, Z. 2010. Motion fields for interactive character locomotion. *ACM Trans. Graph. 29,* 6 (Dec.), 138:1–138:8.

LEVINE, S. AND KOLTUN, V. 2013. Guided policy search. In *ICML '13: Proceedings of the 30th International Conference on Machine Learning*.

LEVINE, S. AND KOLTUN, V. 2014. Learning complex neural network policies with trajectory optimization. In *ICML '14: Proceedings of the 31st International Conference on Machine Learning*.

LIU, L., YIN, K., AND GUO, B. 2015. Improving Sampling-based Motion Control. *Computer Graphics Forum 34,* 2.

LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph. 31,* 6, Article 154.

LIU, L., YIN, K., VAN DE PANNE, M., SHAO, T., AND XU, W. 2010. Sampling-based contact-rich motion control. *ACM Trans. Graph. 29,* 4, Article 128.

LIU, L., YIN, K., WANG, B., AND GUO, B. 2013. Simulation and control of skeleton-driven soft body characters. *ACM Trans. Graph. 32,* 6, Article 215.

MACCHIETTO, A., ZORDAN, V., AND SHELTON, C. R. 2009. Momentum control for balance. *ACM Trans. Graph. 28,* 3.

MORDATCH, I., DE LASA, M., AND HERTZMANN, A. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph. 29,* 4 (July), 71:1–71:8.

MORDATCH, I. AND TODOROV, E. 2014. Combining the benefits of function approximation and trajectory optimization. In *Proceedings of Robotics: Science and Systems*. Berkeley, USA.

MORDATCH, I., TODOROV, E., AND POPOVIĆ, Z. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph. 31,* 4 (July), 43:1–43:8.

MUICO, U., LEE, Y., POPOVIĆ, J., AND POPOVIĆ, Z. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph. 28,* 3.

MUICO, U., POPOVIĆ, J., AND POPOVIĆ, Z. 2011. Composite control of physically simulated characters. *ACM Trans. Graph. 30,* 3 (May), 16:1–16:11.

PENG, X. B., BERSETH, G., AND VAN DE PANNE, M. 2015. Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics (to appear)*.

PETERS, J. AND SCHAAL, S. 2007. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. ACM, New York, NY, USA, 745–750.

PETERS, J. AND SCHAAL, S. 2008. Reinforcement learning of motor skills with policy gradients. *NEURAL NETWORKS 21,* 4 (MAY), 682–697.

POPOVIĆ, Z. AND WITKIN, A. 1999. Physically based motion transformation. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 11–20.

RAIBERT, M. H. AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. In *ACM SIGGRAPH Computer Graphics*. Vol. 25. ACM, 349–358.

ROSS, S., GORDON, G., AND BAGNELL, J. A. D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artifical Intelligence and Statistics (AISTATS)*.

SOK, K. W., KIM, M., AND LEE, J. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph. 26,* 3, Article 107.

SULEJMANPAŠIĆ, A. AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Transactions on Graphics (TOG) 24,* 1, 165–179.

TAN, J., GU, Y., LIU, C. K., AND TURK, G. 2014. Learning bicycle stunts. *ACM Trans. Graph. 33,* 4 (July), 50:1–50:12.

TAN, J., LIU, C. K., AND TURK, G. 2011. Stable proportional-derivative controllers. *IEEE Comput. Graph. Appl. 31,* 4, 34–44.

TASSA, Y., EREZ, T., AND TODOROV, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 4906–4913.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Near-optimal character animation with continuous control. *ACM Trans. Graph. 26,* 3 (July).

WAMPLER, K. AND POPOVIĆ, Z. 2009. Optimal gait and form for animal locomotion. *ACM Trans. Graph. 28,* 3, Article 60.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2009. Optimizing walking controllers. *ACM Trans. Graph. 28,* 5, Article 168.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2010. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph. 29,* 4 (July), 73:1–73:8.

WANG, J. M., HAMNER, S. R., DELP, S. L., AND KOLTUN, V. 2012. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph. 31,* 4, 25.

YE, Y. AND LIU, C. K. 2010. Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph. 29,* 4 (July), 74:1–74:9.

YIN, K., COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2008. Continuation methods for adapting simulated skills. *ACM Trans. Graph. 27,* 3, Article 81.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. SIMBICON: Simple biped locomotion control. *ACM Trans. Graph. 26,* 3, Article 105.

ZORDAN, V., BROWN, D., MACCHIETTO, A., AND YIN, K. 2014. Control of rotational dynamics for ground and aerial behavior. *Visualization and Computer Graphics, IEEE Transactions on 20,* 10 (Oct), 1356–1366.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph.*, 697–701.