Simulation for Control

KangKang Yin, kkyin@comp.nus.edu.sg,
National University of Singapore
Libin Liu, libinliu@cs.ubc.ca,
University of British Columbia
Michiel van de Panne, van@cs.ubc.ca,
University of British Columbia

Abstract. Computer simulation is a powerful tool for developing motion controllers in humanoid robotics and physics-based character animation. In this chapter, we first motivate simulation-based methods for the control of physical robots and digital avatars, and discuss the major differences between developing controls in simulation and on real robots. We then detail a series of model-free algorithms developed by our group over the years. These algorithms are based on sampling time-indexed control actions for short duration motion fragments, in order to track desired reference motion trajectories. We term the family of algorithms SAMCON (Sampling-based Motion Control). The basic SAMCON and the improved SAMCON methods reconstruct open-loop controls; the reduced-order feedback policy learning and the guided SAMCON then build robust linear feedback strategies around the open-loop controls. We further introduce a general framework – Control Graphs – that learns and organizes multiple physics-based motion skills and their transitions from a set of example motion capture clips. This offers a potential solution for the development of rich motion repertoires for humanoid robots as well as encourages widespread adoption of physics-based methods for character animation. Finally, we discuss research opportunities for future research on simulation-based control methods and transferring such controls to physical robots.

Keywords: simulation, motion control, control graphs, sampling, policy search, motion capture

1 Introduction

Research on motion control with simulation tools mainly originates from the field of robotics and physics-based computer animation. In robotics, the ever-growing complexity of robots and their environments drives the need for experimentation in simulation rather than directly performing experiments on physical robots [11,12,9]. One obvious reason is that real hardware is expensive and breaks easily, and therefore it is commonly worthwhile to validate control strategies in simulation before applying them on the robots. In addition, digital models are easier to alter, numerical parameters are easier to tune, and simulations generally run faster than realtime. As a result, learning controllers in simulation can potentially boost the performance by orders of magnitude. Furthermore, some

algorithms, including many machine learning methods, require large datasets of state transitions or state action pairs, including failure cases, and these are difficult or impossible to obtain from the physical system.

However in many cases, controllers trained and learned in simulation cannot be directly applied to physical robots, mainly due to hardware constraints and modeling errors. First, physical robots have limited sensing capabilities that are also noisy and delayed; limited actuators that may be noisy and have difficult-to-model properties; and limited onboard capacity for computation. In contrast, simulations allow instant and exact access to the state of the world; actuators are noise and fault-free; and computational resources can be abundant. These problems can be mitigated by deliberately incorporating limitations and noise in the simulated world. Second, the simulated world and the real world simply cannot exactly match each other. The kinematic and dynamic properties of the simulated robots only approximate those of the real robots. The simulated world introduces numerical errors such as discretized integrations and the use of simplified models of real-world physics, such as simplified contact, collision, and friction models that make the forward dynamics tractable. At the same time, the simulated world is also perfect and noise-free in a way that is unrepresentative of the real world. For example, a friction coefficient set to 0.8 will remain exactly 0.8 at all times; yet in real world no surface can guarantee such condition everywhere. Therefore it is helpful to bear in mind all the hardware vulnerabilities and modeling imprecisions, in order to develop robust and adaptive control methods in simulation that will then also work when applied to physical robots.

In physics-based character animation, control methods have been developed through simulation with goals different from those of robotics. It is often sufficient to achieve physically-plausible simulations rather than physically-accurate simulations. Moreover, visual realism and richness are emphasized more than robustness and optimality. For example, a humanoid robot should try to avoid falling at all times, and in extreme cases where a fall is unavoidable, it should fall so as to minimize the damage to the environment and to itself. In contrast, a digital character can choose to fall gracefully rather than seeking to avoid a fall at all costs. The character can also choose to fall in multiple styles, some clumsy, some graceful, some childish, some heroic. Stylish movements are more interesting to look at than optimal and robust ones. With the luxury of having no hardware limits and no hardware that could be damaged during learning, physics-based characters can achieve more visually compelling and aggressive motions than those attainable from physical robots. Our focus in this chapter is therefore mainly on control methods developed in the digital domain and which can therefore take full advantage of the benefits afforded by simulated worlds.

Physics-based animation techniques have been widely adopted for passive physical phenomena and motions, including digital water, smoke and fire, rigid body simulations, and ragdoll simulations (i.e., passive character simulations). These methods are now ubiquitous in visual effects for films and games. However, motion capture remains the prevalent method for achieving high quality character animation, given the difficulty of controlling and simulating the active,

muscle-driven motions of animals and humans. The control difficulties in realizing complex human movements arise from non-linear dynamics, under-actuated systems, and the obscure nature of human control strategies. Nevertheless, the problem of physics-based character control has seen tremendous progress due to ongoing research efforts. Under-constrained motions such as balancing and walking, as well as highly-dynamic skills such as parkour style terrain crossing and gymnastics, can now be simulated and controlled in real time with a motion quality that is nearly indistinguishable from motion capture data.

In this chapter we introduce a series of model-free algorithms based on sampling control actions for short duration motion fragments, in order to track desired reference motion trajectories. We term the family of algorithms as SAMCON (Sampling-based Motion Control), which include the basic SAMCON and the improved SAMCON methods for building open-loop controls. We then present two methods to build robust feedback strategies around the open-loop controls reconstructed by SAMCON. The first learns fixed reduced-order linear feedback policies that are invoked throughout an entire motion episode or motion cycle, while the second learns fragment-level (time-indexed) linear feedback policies. Lastly, we describe Control Graphs, a general framework for learning and organizing multiple motion skills and their transitions with a minimal amount of prior knowledge, in a fashion that is analogous to Motion Graphs for kinematic animations. Once built, control graphs support real-time physics-based simulation of multiple characters, each capable of a diverse range of robust movement skills, including locomotion, highly dynamic kicks and gymnastics, standing, rising motions, and in-between transitions. To begin, we first place our methods within the context of previous work.

2 Related Work

We first briefly discuss the choice of simulation tools and then focus on control methods and algorithms that take advantage of multiple simulations while treating the specific simulator as a black box. We note that in our experience, the specific choice of simulation platforms make little difference when the control methods are sufficiently robust.

Simulation Tools In the 1980s and early 1990s, the common options for forward dynamics simulation of multilinked articulated bodies consisted of a few third-party packages, together with in-house proprietary simulators. Over the years, various research groups have released their relatively mature simulation packages, and a variety of commercial products are also available on the market for developing games and training software. ODE (Open Dynamics Engine) is widely used in research mainly because it is open source, relatively small and well documented. PhysX and Havok mainly target passive physics for gaming and training applications, and are currently supported by NVidia and Microsoft

respectively. Bullet is another influential physics engine widely adopted for simulations in motion pictures and its author was awarded a scientific and technical Academy Award in 2015 for its development. Modern packages usually handle contact dynamics and integrate easily with modeling and visualization packages, in addition to simulating jointed structures.

Generally speaking, in humanoid robotics, simulation tools often choose to work in the joint angle space, i.e., generalized coordinates or reduced coordinates, so that fast and stable recursive dynamics algorithms can be utilized. SD/FAST, MATLAB robotics toolbox, and DART all work with reduced coordinates. For animation and gaming, full coordinates, also known as Cartesian coordinates, are prefered in order to also efficiently support multiple independent rigid bodies in addition to articulated figures. Velocity stepping methods on full coordinates are well suited to handle contacts and collisions but will have constraint violations for joints. MathEngine, ODE, Bullet, PhysX, and Havok all work on full coordinates. Several studies have compared some of these physics engines quantitatively. For example, Bullet, Havok, MuJoCo, ODE and PhysX, have been compared for model-based robotics [9]. Their conclusion is that MuJoCo performs the best in robotics-related tests, while the others win in gaming-related tests. ODE, PhysX, Bullet and Vortex have also been compared for a biped locomotion controller SIMBICON [51,10]. They conclude that if the high-level motion controller is robust enough, the simulations from different physics engines make little difference.

Physics-based Animation Physics-based simulation and control of passive rigid bodies have been quite mature and popular in graphics-driven industries such as the gaming and film industry. The control problems there usually involve generating controllable simulations and plausible variations of passive simulations [2,38,44]. Simulation and control of human-like characters, however, is currently uncommon in industry but remains quite active as a research topic. 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 maneuvers, 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.

There are three broad categories of algorithms for solving the control problem for physics-based characters. The spacetime optimization, or trajectory optimization framework pioneered physics-based animation by imposing dynamics constraints such as equations of motion in optimizing animation trajectories that also satisfy user constraints, such as given keyframes [47,1]. Such methods incur high computational cost and possibly objective function tuning, and do not guarantee physically-plausible controls for successful forward simulations. Recently, control actions, typically joint torques, can be directly optimized for based on high-level control features using hybrid inverse/forward dynamics al-

gorithms [29,5]. Model-based optimal control also provides a general method for developing robust control about given reference trajectories [32].

Controllers without explicit and exact modeling of the dynamics have also been designed with the help of human expertise [15,51,4] or learned through optimization [40,41]. Such controllers can be quite robust if appropriate feedback laws are incorporated, such as the foot placement strategy in SIMBICON-type controllers [51,50]. Data-driven methods that track motion capture examples provide superior realism and variations [20,27] for physics-based human-like characters. In particular, the sampling-based control strategy proposed in [27] has demonstrated the ability to robustly track a wide variety of motions, including those involving complex changing contacts. Therefore in this chapter we will focus on developing robust open-loop and closed-loop controls based on this method. In addition, we present learning methods that enable robust multi-skilled characters.

Sampling-based Methods Our use of sampling is inspired by past successes of sampling-based methods in various fields. In robotics, randomized sampling algorithms for path planning [43,19,18], especially RRTs (Rapidly-exploring Random Trees) offer significant benefits in speed and robustness over conventional deterministic planning algorithms. RRTs have also been used for manipulation planning in character animation synthesis [48]. In this chapter, we reconstruct controls to produce plausible trajectories. A trajectory is a path with a time constraint. Thus trajectory planning is a more difficult task than path planning: while path planning only needs to consider kinematic constraints such as collisions with static objects, trajectory planning has to take dynamic constraints into account as well.

In computer vision, visual tracking with Sequential Monte Carlo methods and particle filtering algorithms [7,8,16] have been quite successful. In particle filtering, distributions are represented by a set of particles; each particle has a likelihood weight assigned to it that represents the probability of that particle being sampled from the probability density function. To prevent sample depletion due to weight collapse during particle propagation, a resampling step adapts the distribution of particles according to their likelihoods. Similarly, in the basic SAMCON, we only generate samples for the next time step from good samples in the current time step.

Sampling-based algorithms have also been explored in both passive animation [2,44] and active animation [40]. The use of local stochastic search or genetic algorithms with built-in randomness can produce interesting motion, morphology, and behavior variations. More recently, sampling-based optimization methods, such as the CMA (Covariance Matrix Adaptation) strategy, have been shown to be effective in optimizing walking controllers [46] and generating optimal gaits and morphologies for animal locomotion when combined with traditional derivative-based spacetime optimization [45]. The improved SAMCON algorithm also utilizes a special CMA method referred as (μ_W, λ)-CMA-ES [14].

Policy Search 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., flexible navigation [20], and for physics-based models, e.g., terrain traversal with constraints [3] and with highly dynamic gaits [33]. Policy search methods are often applied to problems having continuous action spaces, often searching the parameter space using stochastic optimization algorithms such as EM-based approaches [35], policy gradient [36], and reinforcement learning with reward-weighted regressions [34]. 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 numbers of samples, and convergence to local optima. Thus adapting such methods to highly agile human motions other than basic walking and running remains an open problem.

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 approximation, which then informs the collection of new samples, and so forth, e.g., [21,22,31]. The guided learning pipeline of control graphs that we will describe later has a similar guided-learning structure but is unique in various ways, thus more powerful than previous methods to date. 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.

3 Sampling-based Motion Control

We introduce a family of sampling-based algorithms for building both open-loop and closed-loop controls from motion examples. We start with defining some terms and symbols we will frequently use hereafter. In a multibody system, i.e., a digital avatar, we define a *pose* as the aggregation of all the internal joint angles and the root orientation and position at a particular time. A *sample* is defined by a pose displacement, that when added to a pose, forms a new pose. For convenience, we also use *sample* to refer to the new displaced pose. A pose or sample only contains positional information. For a dynamically simulated system, we also need to consider the *state*, which contains both position and velocity information of the system. Table 1 lists the various symbols we use throughout this chapter. Note that the indices i, j, k can have other meanings depending on the context.

All simulations are performed with the open-source Open Dynamics Engine (ODE). Our system can simulate four interacting characters in real-time or a single character in $10\times$ real-time on a mid-range desktop with our single threaded C++ implementation. Except for the retargeting experiments, all our experi-

Symbol	Description
$\boldsymbol{p}$	pose
$\hat{\boldsymbol{p}}$	target pose for PD-servos
$\Delta\hat{\boldsymbol{p}}$	offset on target poses
$\boldsymbol{m}$	simulated motion clip
$\tilde{\boldsymbol{m}}$	input 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
$\boldsymbol{\pi}$	control 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	index for reconstruction trial or policy search iterations
j	sample index for guided SAMCON
k	index for control fragments

Table 1. Symbols

ments are performed with a human model that is $1.7m$ tall and with a mass of $62kg$. It has 45 DoFs (Degrees of Freedom) in total, including 6 DoFs for the position and orientation of the root. Where possible we use the same PD-gains for all the joints. For basic locomotion, we can simply set $k_p = 500$, $k_d = 50$ (or similar values) for each joint. For highly dynamic stunts, a stronger waist, e.g., $(k_p = 2000, k_d = 100)$, and leg joints, e.g., $(k_p = 1000, k_d = 50)$, are necessary.

3.1 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 internal degree of freedom with PD-servos:

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

where q and $\dot{q}$ represent the joint rotation and rotational velocity respectively, and the tracking target $\hat{q}$ is given by a target pose $\hat{\boldsymbol{p}}$ derived from a target trajectory $\hat{\boldsymbol{m}}$. For the basic SAMCON algorithm, we used a simulation time step of $0.5ms$ with the conventional PD servos. For the later sections, we follow the idea of Stable-PD control [42] and replace the second term of Equation 1 with implicit damping [28] for better stability. This allows us to use a large

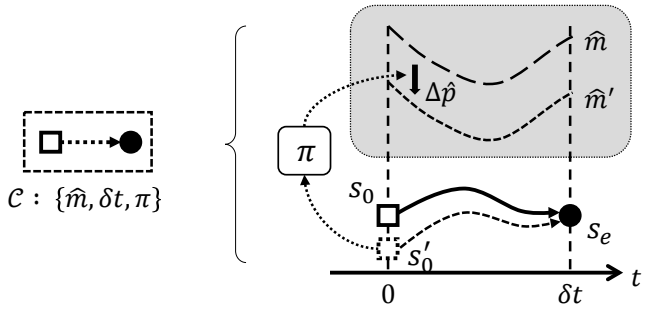


Fig. 1. A control fragment: when the simulate state $\boldsymbol{s}_0'$ drifts away from the reference start state $\boldsymbol{s}_0$, the control policy $\boldsymbol{\pi}$ is involved to compute a compensation $\boldsymbol{\Delta p}$ that offsets the reference clip $\hat{\boldsymbol{m}}$ to $\hat{\boldsymbol{m}}'$. By tracking $\hat{\boldsymbol{m}}'$ with PD-servos, the simulation can end near the reference end state $\boldsymbol{s}_e$ in δt seconds.

Fig. 2. A chain of control fragments form a controller.

simulation time step of $5ms$, which significantly speeds up the learning process and improves the online performance.

Naive tracking of a reference trajectory with joint-level PD-servos is typically unsuccessful for several reasons. Tracking the motions of individual joints provides no ability to track or restore the overall balance of a biped model. The simplistic nature of local PD-controllers also result in a reactive model of control that has no mechanisms for motion or force anticipation. The reference motions themselves may be physically infeasible and the kinematic and dynamic properties of the simulated biped model may also differ from that of the motion capture subjects. Lastly, there are modeling errors associated with a rigid body simulator, including oversimplified rigid biped models and simplified contact and collision models.

When tracking a walking motion, the virtual character usually falls within one or two steps. When directly tracking a sideways roll, the character cannot roll more than 40 degrees. It is thus necessary to add corrections to the reference trajectory in order to complete the underlying tasks successfully. Hereafter we refer to the process of computing such corrections as control reconstruction, whose output is a tracking-based motion controller that can generate appropriate corrections at run time to lead the simulated characters to perform the reference motion tasks successfully.

Control Fragments The basic units of the controllers in our framework, represented by the symbol $\mathcal{C}$, are referred to as control fragments. More specifically,

a control fragment is a tuple $\{\delta t, \hat{\boldsymbol{m}}, \boldsymbol{\pi}\}$ as indicated in Figure 1, where $\hat{\boldsymbol{m}} = \hat{\boldsymbol{p}}(t)$ represents 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 $\delta t \approx 0.1$ seconds. 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 modeling errors, numerical drift, and perturbations.

The control actions are represented in the pose space rather than in torque space; the low-level open-loop nature of torques is likely to make motions more sensitive to dynamics modeling errors, as might be expected in transferring controllers to real robots, to different characters, or when using different simulators. The control policy, $\boldsymbol{\pi}$, is therefore used to compute a corrective action $\boldsymbol{a}$ to produce an offset $\Delta\hat{\boldsymbol{p}}$, that is added to $\hat{\boldsymbol{m}}$ in order to eliminate the deviation. As illustrated in Figure 1, this offset remains fixed during the duration of a 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 near the desired end state, $\boldsymbol{s}_e$. Here the operator $\oplus$ represents a collection of quaternion multiplications between corresponding joint rotations. The general form of the control policy for each control fragment can be written as:

$$\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s}; \boldsymbol{\theta}_k) \tag{2}$$

where $\boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\theta})$ represents the probability density of the action $\boldsymbol{a}$ given the state $\boldsymbol{s}$ and policy parameters $\boldsymbol{\theta}$, and $\boldsymbol{s}$ and $\boldsymbol{a}$ are vectors representing the simulation state and action, respectively.

We use a selected subset of state and action features in order to facilitate a compact control policy. For most of the skills developed in this chapter, 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 converted into a quasi-static 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 an 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 now be formally defined as a cascade of control fragments $\mathcal{W} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N\}$, as depicted in Figure 2, which can be executed to reproduce a given reference motion clip.

3.2 Basic SAMCON

We now detail the basic sampling-based motion control (SAMCON) method that reconstruct controls from a reference input motion capture clip $\tilde{\boldsymbol{m}}$. The core idea

Algorithm 1 SAMCON

Require:
a chain of control fragments $\mathcal{W} = \{\mathcal{C}_k\}, k = 1, \ldots, N$
the start state $\boldsymbol{s}_0$
Ensure: a successful execution of the sequence $\boldsymbol{\tau}$

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

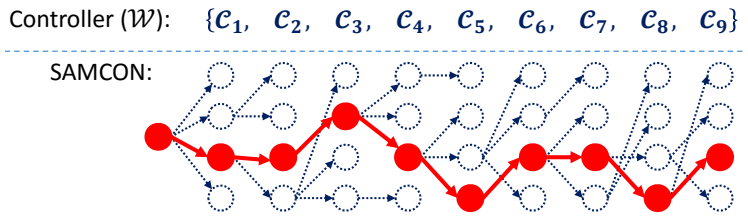


Fig. 3. Schematic illustration of the SAMCON process, with $N_s = 4$ and $n_s = 2$. All the samples for all the control fragments form a tree. The final reconstructed controller is colored in red.

is to draw a large number of samples, i.e., pose displacements for each control fragment, $\mathcal{C}_k$, in order 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. A straightforward way to draw samples is from normal distributions that have fixed mean and covariance and are state-independent [27], e.g., $\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.

Algorithm 1 outlines the main steps of SAMCON, and Figure 3 provides a simple example. 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 the fixed normal distribution as described above. 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 $\boldsymbol{a}_1^j$. 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 by a sample cost function to measure the goodness of a sample.

After executing all N_s sample actions and obtaining the simulation tuples $\{\tau_1^j\}$, SAMCON selects and saves the n_s best tuples $\{\tau_1^{j*}\}$, as measured by the lowest sample 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 chain is reached. Finally, the resultant controller $\boldsymbol{\tau} = \{\tau_k\}$ is chosen to be the best path of all saved tuples $\{\tau_k^{j*}\}$.

Sample Cost Evaluation Our cumulative sample cost function uses a weighted sum of the terms given in Equation 3, which measures the dissimilarity between a simulation tuple's end state and that of its corresponding reference end state.

$$\begin{aligned} E = w_p E_p + w_r E_r + w_e E_e + w_b E_b \\ + w_c E_c + w_v E_v + w_L E_L + w_a E_a \end{aligned} \tag{3}$$

where the terms for pose control E_p, root control E_r, end-effector control E_e, and balanced control E_b will be explained shortly. 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 $(w_p, w_r, w_e, w_b, w_c, w_v, w_L, w_a) = (4.0, 4.0, 10.0, 1.0, 3.0, 0.1, 0.03, 0.05)$ for all our experiments. Our results are not sensitive to the exact values of these weights so other values of the same order of magnitude can be used as well. Now we describe the details of the different terms. Note that over the years these terms have been evolved to handle a greater variety of motions so they appear slightly different here from their first definition in Liu et al. [27].

Pose Control: We favor states that are close to the trajectory. Here we simply use a weighted squared distance of internal joint angles and angular velocities.

$$E_p = \frac{1}{\sum w_i} \sum w_i (d_q(\boldsymbol{q}_i, \tilde{\boldsymbol{q}}_i) + 0.1 * d_v(\dot{\boldsymbol{q}}_i, \tilde{\dot{\boldsymbol{q}}}_i)) \tag{4}$$

where $d_v(\dot{\boldsymbol{q}}, \tilde{\dot{\boldsymbol{q}}}) = \|\dot{\boldsymbol{q}} - \tilde{\dot{\boldsymbol{q}}}\|_2$ is the Euclidean distance between two vectors, $d_q(\boldsymbol{q}, \tilde{\boldsymbol{q}}) = \|\log(\boldsymbol{q} \bullet \tilde{\boldsymbol{q}}^{-1})\|_2$ is the distance between two quaternions, and $\bullet$ represents quaternion multiplication. w_i adjusts the relative importance of different joints. We usually use $w_i = 5.0$ for the waist and the leg joints, and $w_i = 1.0$ for the rest of the joints. We can adjust the weights to produce further variations, if desired.

Root Control: The root orientation of underactuated systems can only be controlled indirectly via potentially-complex interactions between body parts and the ground. We therefore select samples that closely track the orientation of the reference trajectory, as captured by:

$$E_r = d_q(\boldsymbol{q}_0, \tilde{\boldsymbol{q}}_0) + 0.1 * d_v(\boldsymbol{q}_0, \tilde{\dot{\boldsymbol{q}}}_0) \tag{5}$$

End-effector Control: Many motions involve non-trivial interactions between the ground and the hands and feet. The locations of these end-effectors are

thus crucial to task success. We use a term that monitors the error between the desired and current height of end-effectors, which ensures foot clearance during locomotion for instance:

$$E_e = \frac{1}{k}\sum(h_i - \tilde{h}_i)^2, \tag{6}$$

where k is the total number of end-effectors considered, h_i is the height of the i^{th} end-effector.

Balance Control: Balance control is typically achieved by adjusting the Center of Mass (CoM) with respect to the support polygon. We use the relative position of the CoM with respect to each end-effector instead. This has two advantages. First, we do not need to detect support polygons from noisy captured motions. Second, even when an end-effector is not in contact with the ground, its relative position with respect to the CoM still counts. This is important for the end-effector to prepare for a proper landing position.

We calculate the balance deviation according to:

$$E_b = \frac{\sum w_i d_v(\boldsymbol{r}_{ci} - \tilde{\boldsymbol{r}}_{ci})}{\sum w_i} + 0.1 * d_v(\boldsymbol{v}_{CoM}, \tilde{\boldsymbol{v}}_{CoM}) \tag{7}$$

where $w_i = 1.0/(0.001 + h_i)$, h_i is the height of the end-effector, and $\boldsymbol{r}_{ci} = (\boldsymbol{p}_{CoM} - \boldsymbol{p}_i)$ is the vector from end-effector i to the CoM.

Results Despite its apparent simplicity, the basic SAMCON algorithm can produce successful control reconstructions for relatively short reference motions with rich ground contacts. Figure 4 top shows a simulated forward roll controlled by the reconstruction of the basic SAMCON. Such simulations have been difficult to achieve with other methods due to the many transient contacts that are hard to model.

The inherent robustness of the sampling approach enables straightforward physically based motion transformation and motion retargeting. For example, we can reconstruct the forward rolling onto a $100cm$ height drop as shown in Figure 4 bottom. We can retarget various motions captured from human subjects, such as the barrel roll, to the Asimo-like robot model as shown in Figure 5. These retargeting tasks are challenging for the Asimo model because of the large differences between the human subjects and the Asimo model, in terms of their kinematic parameters, dynamic parameters, and collision detection geometries. Nonetheless, SAMCON succeeds in reconstructing controls for most of the motions we tried for the Asimo-like model. Compared with control reconstruction for a human-like model, SAMCON requires larger sampling windows, produces motions of lower quality, and is less accurate in terms of trajectory tracking. For example, due to wider and boxy legs, and the lack of a waist joint, Asimo cannot twist its spine or its legs around each other like humans do, and has to rely on a shoulder strategy to roll sideways. The Asimo running is sluggish, mostly because Asimo has bent knees defined for its T-pose, which causes early touchdown of the feet when the knees extend during locomotion. We were not

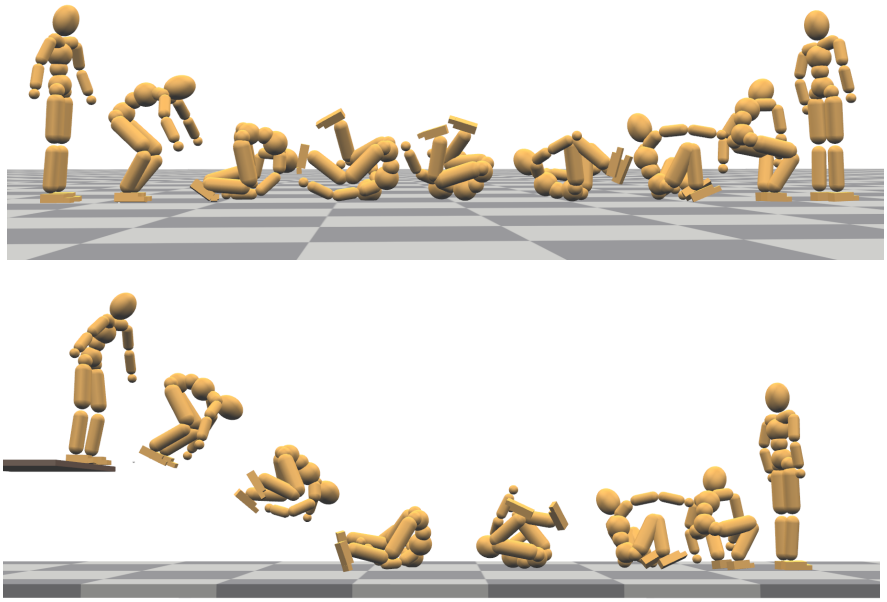


Fig. 4. Top: a simulated forward roll controlled by the reconstruction of the basic SAMCON algorithm. Bottom: the same forward roll transformed to a dive roll.

able to successfully reconstruct controls for the backward roll on Asimo. However, we can produce a successful backward roll from the sampled controls when we give Asimo a 3-DOF neck.

3.3 Improved SAMCON

The basic SAMCON, Despite the success described above, the basic SAMCON method has limitations. Firstly, it samples from fixed distributions and the control reconstruction process treats each trial independently. If failure occurs, the reconstruction restarts with the same sample distributions without learning anything from past experience. Thus when working with challenging motions where the success rate of sampling is low, the method becomes inefficient and requires an excessive number of reconstruction passes. We address this limitation with the key insight that later control reconstruction passes should learn from earlier successes and failures in order to draw better samples. We realize such learning through adaptation of the distributions that we draw samples from. We implement this sample distribution adaptation scheme by a modified CMA (Covariance Matrix Adaptation) method that will be described shortly.

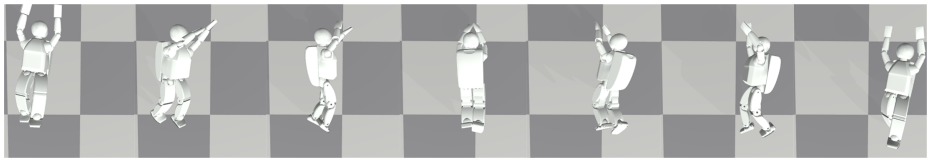


Fig. 5. A sideways roll captured from human retargeted to an Asimo-like robot using the basic SAMCON.

The second limitation of the basic SAMCON occurs because of the need to restart the search from the beginning of the motion when failure occurs. As a result, the method takes excessively long to reconstruct long motions or motions that contain specific events that are critical to its success. For example, motions with long-flight phases such as gymnastic movements need to be controlled more precisely before the take-off, as the flight phase offers limited opportunities to correct errors. We address this issue by integrating a sliding window mechanism into the sample distribution adaption process to improve the reconstruction efficiency and robustness for long motions and motions with critical instants. The sliding window scheme can also prevent overfitting of earlier control fragments by moving them out of the window once they converge.

Lastly, the basic SAMCON is inherently noisy and thus the resulted motions may look jerky. Such artifacts are not so noticeable for floor interactions but can become disturbing for balancing and locomotion tasks. For instance, the head and body of the virtual character may shake unrealistically during standing or while walking. We propose two averaging methods to reduce the noise in reconstructed controls, which are both effective in terms of noise reduction, but the simple averaging method needs multiple control trajectories and may become inefficient for challenging motions. In contrast, the elite averaging method averages elite samples during reconstruction is more efficient in achieving good results.

Sample Distribution Adaptation To equip the original algorithm with a learning ability, we employ the idea of distribution adaptation. The key is to acknowledge that even failed reconstruction passes may still contain good samples in the early iterations. We can thus learn from these elite samples to reshape the default normal distributions from which we draw samples for the next reconstruction trial. We move the centers of the distribution as well as update the shape of the distribution, $\boldsymbol{\pi}_k^i(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) = \mathcal{N}(\hat{\boldsymbol{a}}_k^i, \boldsymbol{\Sigma}_k^i)$, where k is the control fragment index and i is the trial index, to favor samples that are closer to the true solution space, similar to CMA-based algorithms.

More specifically, we adapt the sample distributions for earlier successful control fragments $\mathcal{C}_k$ before we start the next trial T_i, as illustrated in Figure 6. The first trial T_1 draws samples from default normal distributions $\boldsymbol{\pi}_k^1 = \mathcal{N}(\hat{\boldsymbol{a}}_k^1, \boldsymbol{\Sigma}_k^1)$,

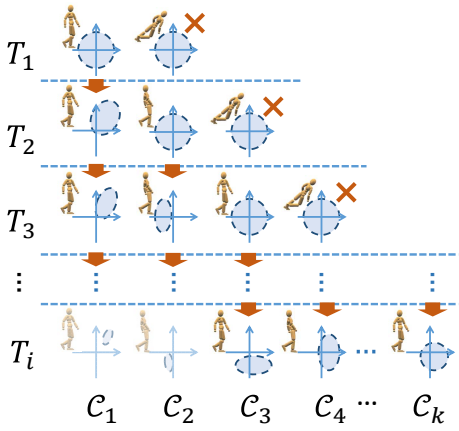


Fig. 6. An illustration of our sample distribution adaptation method. T_1: the first control reconstruction trial fails at control fragment $\mathcal{C}_2$, so only the distribution for $\mathcal{C}_1$ is updated. T_2 and T_3: after a few trials, more distributions get updated and longer control trajectories can be constructed. T_i: the distributions of $\mathcal{C}_1$ and $\mathcal{C}_2$ have converged, the reconstruction window is slid to start from $\mathcal{C}_3$.

where $\hat{\boldsymbol{a}}_k^1 = \mathbf{0}$ and $\boldsymbol{\Sigma}_k^1$ is initialized with the default sampling window size. When T_1 fails at control fragment $\mathcal{C}_2$, we update the sample distribution for $\mathcal{C}_1$ using the elite samples, i.e., the saved samples, and denote the new distribution as $\boldsymbol{\pi}_1^2 = \mathcal{N}(\hat{\boldsymbol{a}}_1^2, \boldsymbol{\Sigma}_1^2)$. Then we start another reconstruction pass T_2 for which we draw samples from the updated distribution $\boldsymbol{\pi}_1^2$ for $\mathcal{C}_1$. This process continues and the algorithm utilizes elite samples of past trials to gradually direct future sample drawings closer and closer to the true solution. This is in contrast to the basic SAMCON, which always draws samples randomly from fixed distributions.

We use the (μ_W, λ)-CMA-ES method [14], a stochastic global optimization technique, to update sample distributions for each iteration. Our algorithm thus only has one new parameter compared to the basic SAMCON: the step size parameter for the CMA algorithm, and we initialize it to 0.1 for all the examples shown. The normal distributions are updated according to the quality of elite samples, that is, better elite samples are weighted more during the update. The sample quality is measured by: a) the height of the sample's offspring subtree; and b) the cumulative cost of the best path from the sample to the leaves.

Note that our distribution adaptation method is different from other applications of CMA in the control optimization literature [1]. These sample from the full control space, which usually results in high computational cost even for sparsely placed control points on the time line, yet we update distributions for each control fragments independently.

Sliding Window The distribution adaptation method described above further enables two improvements to achieve more effective control reconstruction, especially for long motion clips. First, we can choose to run up to a fixed number of control fragments in each trial, rather than wait until the trial fails, so that the sample distributions can be updated in a timely fashion. Second, the distributions for the earlier control fragments will likely converge after multiple updates, thus we can skip these fragments in future trials to focus sampling only

for the later fragments. These two variations combined suggest a sliding window mechanism for control reconstruction.

More specifically, we use fifty control fragments (five seconds) as the size of the sliding window in all our experiments. We slide a fragment outside of the window if: (a) the distribution of the fragment has been updated for at least five times and at most twenty times; and (b) the distribution has stabilized, i.e. the cost of the control has not decreased for the last five trials; or (c) the best sample drawn from the distribution is good enough according to the cost function. Note that long motions usually consist of both high dynamic segments and static balancing periods, thus we need to normalize the cost function according to the kinetic energy of the window to more accurately estimate convergence.

Generally speaking, reconstructing controls for a challenging motion with the basic SAMCON may fail even when using large sampling windows, large number of samples, and multiple passes of reconstruction. In contrast, the improved SAMCON can learn from earlier trials to advance to success window by window. For motions that can be reconstructed successfully by both algorithms, the reconstructed controls from the improved SAMCON are superior in quality. This is because the improved method is able to gradually move and reshape the sample distributions so that smaller sampling windows can be used to reduce sampling noise.

Noise Reduction Sampling-based methods are inherently noisy. Post-smoothing of the controllers does not work as it results in physically invalid controls. We thus follow the law of large numbers, which suggests simple averaging as an effective method of noise reduction. The basic SAMCON can employ averaging in a straightforward fashion. We simply run the reconstruction multiple times until N control trajectories $\hat{\boldsymbol{m}}'_i$, $i = 1 \dots N$ are generated. We denote their corresponding simulated trajectories as $\boldsymbol{m}_i$. Then we average these trajectories as:

$$\bar{\hat{\boldsymbol{m}}}' = \frac{1}{N} \sum_{i=1}^{N} \hat{\boldsymbol{m}}'_i, \qquad \bar{\boldsymbol{m}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{m}_i \tag{8}$$

Note that these trajectories are a collection of poses in time, so the averaging operation is applied to all the frames at the same instant of time on the trajectories. Since time scaling may occur during control reconstruction, the trajectories need to be first resampled to the same length in time before they can be averaged.

We then rerun the sampling algorithm using $\bar{\hat{\boldsymbol{m}}}'$ as the initial solution and $\bar{\boldsymbol{m}}$ as the reference. This final reconstruction pass only needs to use very small sampling windows, and thus a much smoother control trajectory can be obtained. Note again that $\bar{\hat{\boldsymbol{m}}}'$ itself is not a control solution, just like any kinematically smoothed version of $\hat{\boldsymbol{m}}$, because they violate the equations of motion constraint. However, $\bar{\hat{\boldsymbol{m}}}'$ is obtained from averaging physically plausible controls and is thus closer to the solution than kinematically smoothed controls.

The averaging method described above is simple, effective, and can be directly applied to the basic SAMCON. However, it requires multiple passes to obtain at

least N successful control trajectories for the averaging operation. If the success rate of control reconstruction is low, for example for challenging motions that motivated the distribution adaptation method, the computation can become prohibitive. We now introduce *Elite Samples Averaging*, which reduces noise by using the elite samples as soon as the first control trajectory $\hat{\boldsymbol{m}}_1$ is obtained.

More specifically, for each control fragments in $\hat{\boldsymbol{m}}_1$, we compute a weighted average of all the elite samples $\Delta\hat{\boldsymbol{p}}_k^j$ as follows:

$$\overline{\Delta \boldsymbol{p}_k} = \frac{\sum h_j \Delta \hat{\boldsymbol{p}}_k^j}{\sum h_j} \tag{9}$$

where h_j is the sample's subtree height in Figure 3 and has to be larger than four for a sample to be considered elite. It measures the quality of samples by their descendants, and is therefore better than the near-sighted sample cost function. We then use the averaged samples as an improved initial guess for another control reconstruction pass.

We also shrink the sampling windows gradually and replace the reference motion with the simulated motion periodically. This is because the average of elite samples is a better initial guess for control reconstruction, and the simulated motion is a dynamically-filtered version of the reference motion. For all our experiments, we reduce the sampling window by 30% after each successful trial. The algorithm is not sensitive to this parameter to be successful, and will simply take longer if less reduction is used. We replace the reference motion with the simulated motion of the last successful reconstruction whenever we encounter a failure in the reconstruction process. The above described scheme learns from past successful trials as fast as possible rather than wait for multiple successful reconstructions. Therefore, the sampling noise can be reduced much faster as compared to the simple averaging scheme.

Results We have tested the improved SAMCON algorithm using a variety of motions, including low-momentum motions, high-dynamic motions, airborne motions, and long performance routines. We initialize the sampling window to 0.1, which is then gradually adapted after each trial. Figure 7 shows the animation strips of a gymnastic flip on the top, a stylized walk in the middle, and the stylized walk retargeted to a monster character at the bottom.

In summary, the improved SAMCON using the sample distribution adaptation method coupled with the sliding window scheme is more successful for challenging motions, requires less parameter tuning, and produces motions of better quality, compared with the basic SAMCON. We refer the interested readers to [25] for more details. The simulated motions are generally of high quality, but may still contain visible noise for motions with long static balancing periods where the absolute value of visually tolerable noise level is low. In such cases, we can apply the elite averaging method to further reduce the noise level in a few additional control reconstruction runs. When computational resources are abundant, simple averaging is preferable for better noise reduction effect.

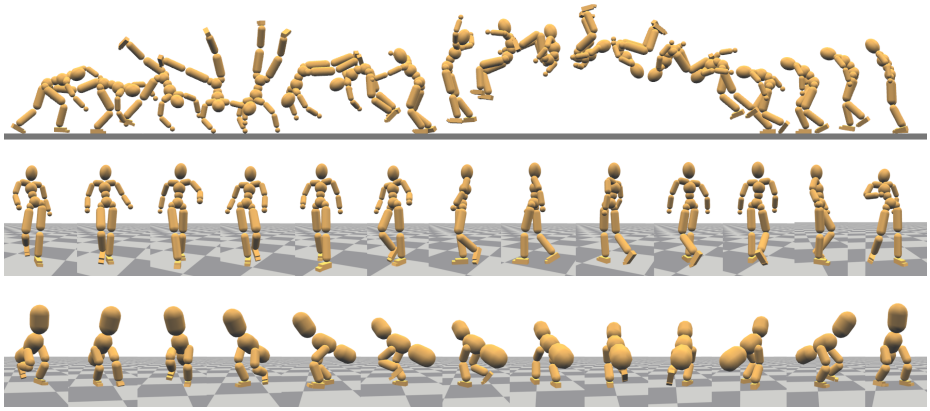


Fig. 7. Simulations of a gymnastic flip (top), a stylized walk (middle), and the stylized walk retargeted onto a monster character (bottom).

3.4 Learning Feedback Policies

The control fragments reconstructed thus far by both the basic SAMCON and the improved SAMCON can only produce open-loop controls, meaning that the simulated characters can complete the tasks successfully without external perturbations, as the reconstructed control policies $\boldsymbol{\pi}$ are strictly functions of time. In order to produce robust execution of skills that can recover from a reasonable amount of environmental or user-input perturbations, closed-loop feedback policies are necessary. We explore linear feedback policies as follows:

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{\pi}(\boldsymbol{s};\boldsymbol{\theta}) \\ &= \boldsymbol{\pi}(\boldsymbol{s};\boldsymbol{M},\hat{\boldsymbol{a}}) \\ &= \boldsymbol{M}\boldsymbol{s} + \hat{\boldsymbol{a}} \end{aligned} \tag{10}$$

where $\boldsymbol{M}$ represents a feedback gain matrix, and $\hat{\boldsymbol{a}}$ is an affine term similar to the mean of the Gaussian control policy in the improved SAMCON method. The feedback parameters are then defined as $\boldsymbol{\theta} = (\boldsymbol{M}, \hat{\boldsymbol{a}})$.

Learning one linear feedback policy for each control fragment results in a high-dimensional control parameterization, $\boldsymbol{\theta} = \{\boldsymbol{\theta}_k\} = \{(\boldsymbol{M}_k, \hat{\boldsymbol{a}}_k\}, k \in \{1, \dots K\}$. Learning through policy search in this parameter space is challenging. We provide two alternative approaches to this problem. The first is to reduce the dimensionality before learning. We can learn feedback laws not at the granularity of the control fragments level, but at the motion clips level. We can further learn reduced-order feedback laws which factorizes the feedback gain matrix $\boldsymbol{M}$ to reduce its dimensionality. These measures reduce the flexibility and robustness of the resultant controls, but nevertheless are quite successful for a significant

number of cyclic motion tasks. We refer to this method as learning *reduced-order feedback policies*.

The second approach, which we refer to as guided SAMCON, retains the dimensionality of the control space but builds on the recent ideas of *guided policy search* algorithms. Guided SAMCON is an iterative policy search process where new samples from the control oracle, i.e., SAMCON, inform the construction of an improved policy, which then informs the collection of new samples by the control oracle.

We will now discuss learning reduced-order feedback policies for individual motion skills and defer the discussion on guided SAMCON and control graphs to Sections 3.5 and 3.6.

Reduced-order Feedback Structure We rewrite the above general affine form of linear feedback policies as a linear function of changes in sensory observations:

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{M}\boldsymbol{s} + \hat{\boldsymbol{a}} \\ &= \boldsymbol{M}(\boldsymbol{s} - \hat{\boldsymbol{s}}) \\ &= \boldsymbol{M}_F \cdot \boldsymbol{\delta s} \end{aligned} \tag{11}$$

where $\boldsymbol{M}_F$ is the same $m \times n$ feedback matrix as $\boldsymbol{M}$ but we use the subscript F to emphasize its full-order characteristics, which will then be factorized into reduced-order feedback matrices shortly. The reference sensory observations, $\hat{\boldsymbol{s}}$ can be obtained from the nominal open-loop control policy, such as the output of SAMCON. Here we only study *static output feedback*, where the feedback gains $\boldsymbol{M}_F$ do not change over time. $\boldsymbol{M}_F$ either stays fixed throughout the entire motion skills, such as running or rolling, or changes only across distinctive phases of motion skills, such as vaulting or drop-rolling.

The full-order linear feedback policy is parameterized by the $m \cdot n$ elements of $\boldsymbol{M}_F$. In order to define more compact policies, we can factor $\boldsymbol{M}_F$ into two components: (i) a $r \times n$ sensory projection matrix $\boldsymbol{M}_{sp}$ that projects high-dimensional sensory observations to a reduced-order state space; and (ii) a $m \times r$ action projection matrix $\boldsymbol{M}_{ap}$ that maps the reduced-order state back to the full action space to produce the feedback compensation. The feedback policy then becomes:

$$\boldsymbol{a} = \boldsymbol{M}_{ap} \cdot \boldsymbol{M}_{sp} \cdot \boldsymbol{\delta s} \tag{12}$$

The reduced order feedback policy has $r(m + n)$ parameters. Choosing $r < mn/(m + n)$ guarantees a policy with fewer parameters than the full policy. This further implies $r < \min(m, n)$. The intermediate reduced-order space of dimension r can be thought of as a latent space defined by a small number of abstract *composite variables* that are particularly useful for providing feedback. We use **(n:r:m)** to describe a linear feedback policy with n-dimensional sensory state, r-dimensional reduced-order space, and m-dimensional actions. Full-order feedback policies will be denoted by **(n:F:m)**.

Additional sparsity can be enforced by encouraging rows and columns to be zero. This can implicitly perform feature selection among actions (rows of M_{ap} set to zero) and sensory observations (columns of M_{sp} set to zero). This is implemented as part of the policy optimization process, as we discuss next.

Policy Optimization We apply policy search using repeated rollouts in order to optimize the linear feedback structures $\boldsymbol{M}$, which consists of either the full matrix $\boldsymbol{M}_F$ or its reduced-order factored form $\boldsymbol{M}_{ap} \cdot \boldsymbol{M}_{sp}$. Given a desired motion task, a cost function is defined. These share a common structure:

$$cost(\boldsymbol{M}) = w \cdot [S(\boldsymbol{M}), E(\boldsymbol{M}), U(\boldsymbol{M}), R(\boldsymbol{M})] \tag{13}$$

The function score is a weighted sum of four terms: $S(\boldsymbol{M})$ rewards structures that make the motion as robust as possible; $E(\boldsymbol{M})$ measures how well the resulting motion meets the environment constraints; $U(\boldsymbol{M})$ measures how well the motion satisfies user specifications; and $R(\boldsymbol{M})$ is an optional regularization term used to encourage the sparsity of $\boldsymbol{M}$ and therefore implicitly feature selection on the sensing and control variables. We use L_1 regularization terms for the norms of column vectors in the sensory projection matrix $\boldsymbol{M}_{sp}$ as well as L_1 norms of row vectors in the action projection matrix $\boldsymbol{M}_{ap}$. This yields:

$$R(\boldsymbol{M}) = w_0 \sum_i \sum_j \left\| \boldsymbol{M}_{sp_{ij}} \right\|_1 + w_1 \sum_i \sum_j \left\| \boldsymbol{M}_{ap_{ij}} \right\|_1 \tag{14}$$

We again use CMA (Covariance Matrix Adaption), a stochastic global optimization technique [14], to optimize the feedback structure. The optimization begins from an initial guess consisting of zero-valued entries. For some control tasks, the optimization is challenging due to the complexity of the dynamical system and the fact that good solutions may only be found in a highly restricted region of the parameter space. For these tasks we therefore break the optimization into multiple stages, each with increasing difficulty, and each using the solution of the previous stage as a starting point.

Results We apply our method of learning feedback policies to a variety of motion skills [6,26], such as running, jumping, and drop-rolling. We will only discuss two of them in details here. For each skill, we detail the sensory variables, control actions, cost functions, and staged-learning (if any). We use the basic SAMCON to construct the open-loop controls from a single captured example of each skill. We test the full-order linear feedback policies as well as several reduced orders.

Rolling: To learn a reduced-order feedback policy for a parkour-style front roll, we repeat a motion captured parkour roll four times to obtain a cyclic kinematic reference motion. We employ the basic SAMCON to construct an open-loop rolling controller that is able to roll the character for one cycle. We further use phase resetting upon right foot contact and right elbow contact to align the

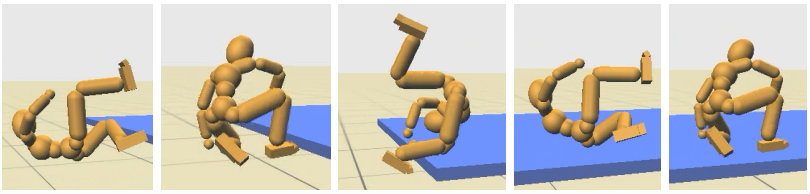


Fig. 8. Forward Rolling with reduced-order linear feedback.

simulation with the reference, rather than adhere to the $0.1s$ simulation duration literally for all control fragments.

Optimization is carried out using a series of sequential stages. Rollouts of 2, 4, 16, and 50 locomotion cycles are used during stages 0,1,2, and 3, respectively. We move CMA to the next stage when the iteration count exceeds 1000 or the value of objective function is smaller than a chosen threshold. The components of the cost function are given by:

$$\begin{aligned} cost(\boldsymbol{M}) &= S(\boldsymbol{M}) + U(\boldsymbol{M}) \\ S(\boldsymbol{M}) &= w_t(N_d T_c - t_{balance}) \\ U(\boldsymbol{M}) &= w_p E_p + w_\tau E_\tau \end{aligned} \tag{15}$$

where N_d is the desired number of rolling cycles, T_c is the length of one reference rolling cycle. The simulation is terminated after a fall or when the desired number of cycles is reached. N_s is the actual number of cycles of the simulation, whose termination time is denoted as $t_{balance}$. E_p computes trajectory deviation between the simulated and the reference motions; and E_τ measures the control energy, which is a sum of all the internal joint torques. We refer the readers to [6] for their exact mathematical definitions. We use $(w_t, w_s, w_p, w_\tau) = (200, 10, 10, 0.005)$ for all the optimization stages.

We learn a successful first-order linear feedback policy (88:1:39) for this skill using full-body state and action descriptions. The full-body state is of 88 dimensions $s = \{h_0, \boldsymbol{v_0}, \boldsymbol{q_0}, \dot{\boldsymbol{q}}_\mathbf{0}, \boldsymbol{q_j}, \dot{\boldsymbol{q}}_\mathbf{j}\}, j \in \{1, \ldots, m\}$. h_0 is the height of the root; $\boldsymbol{v_0}$, $\boldsymbol{q_0}$ and $\dot{\boldsymbol{q}}_\mathbf{0}$ are the linear velocity, the orientation and the angular velocity of the root. These quantities are in the facing coordinate frame of the root. $\boldsymbol{q_j}$ and $\dot{\boldsymbol{q}}_\mathbf{j}$ are the rotation and angular velocity of joint j in the parent body's coordinate frame. The full-body action is PD target pose $a = \{\boldsymbol{q_j}\}, j \in \{1, \ldots, m\}$ of 39 dimensions. With an extra optimization stage with terrain variations, the rolls are robust to these as well. Figure 8 illustrates example rolls onto a step. Developing model-based controllers for this type of task would be difficult because of the rapidly changing ground contacts. In contrast, policy search methods do not need an explicit model of the dynamics; the impact of control is simply observed via policy rollouts. The robustness to pushes varies with respect to the push direction. For example, the character can recover from a large push to the forward right direction (460N), but only a small push to the forward left (220N), both

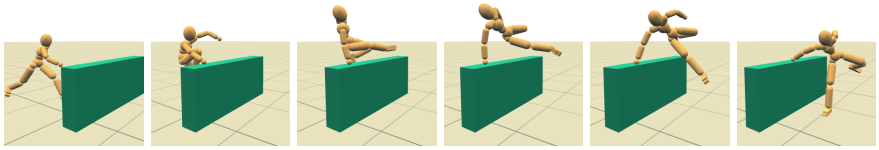


Fig. 9. Vaulting over a 90cm-obstacle with multi-phase reduced-order linear feedback.

last for $0.2s$. This is explained by the asymmetric parkour roll. The results are robust to 10 cm steps in the terrain.

Vaulting: Speed vaulting is a complex skill that consists of multiple contact formations and releases, and full-body clearance over the obstacle, as shown in Figure 9. A single time-invariant feedback matrix for the full course of the maneuver is inadequate to produce robust and realistic controls throughout the motion. Following the common choice of motion decomposition developed by parkour experts, we segment a full vault into three phases: phase 1 - one hand reaches out for the top of the obstacle, and both legs lift off from the ground; phase 2 - the body passes over the obstacle and one foot lands on the other side of the barrier; phase 3 - the character continues to move fluidly in the direction. Phase 1 starts from the first frame and ends when the CoM passes the hand-obstacle contact; phase 2 then starts until one foot touches the ground; phase 3 continues further until the end of the vault. The multi-phase feedback policy for vaulting then becomes $\boldsymbol{a} = \boldsymbol{M}_k \boldsymbol{\delta s}$, where $k \in \{1, 2, 3\}$ is the phase index that correlates with the character state and simulation time.

We experiment with a manually-chosen reduced set of key sensory properties and action parameters. A 15-dimensional $\boldsymbol{s} = \{\boldsymbol{q}_0, \boldsymbol{c}, \dot{\boldsymbol{c}}, \boldsymbol{d}, \boldsymbol{d}_{sw}\}$, where $\boldsymbol{q}_0$ is the root orientation; $\boldsymbol{c}$ and $\dot{\boldsymbol{c}}$ are the CoM position and linear velocity; $\boldsymbol{d}$ is the vector pointing from the COM to the stance foot; and $\boldsymbol{d}_{sw}$ the position of the swing foot with respect to the root, for better control of the clearing height and landing location. In addition, for phase 1 we augment the sensory set with the position of the supporting hand in order to allow for better control of the hand-obstacle contact location. These properties are in the facing frame of the root. We choose the hips and the waist as our key joints for a 9-dimensional action vector, $\boldsymbol{a} = \{\boldsymbol{q}_{swhip}, \boldsymbol{q}_{sthip}, \boldsymbol{q}_{waist}\}$. All these rotations are defined relative to their parent frame.

We again use CMA to optimize for the feedback policy. We only learn the feedback policy for robust transitions between vaulting and running, as vaulting is short in duration and non-cyclic. Several CMA iterations, with the following optimization goal, are already sufficient:

$$cost(\boldsymbol{M}) = E(\boldsymbol{M}) + U(\boldsymbol{M}). \tag{16}$$

Similar to the rolling task, $U(\boldsymbol{M})$ for vaulting contains terms to measure the difference between the simulated motion and the reference, with an emphasis on the end-effector position difference now for precise control of obstacle clearing

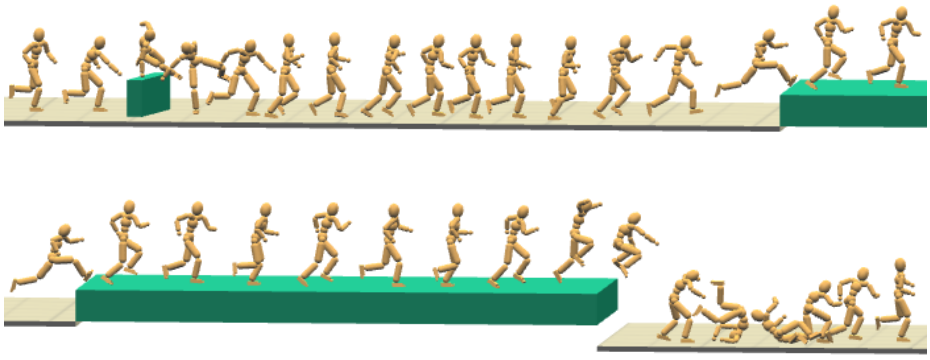


Fig. 10. An physics-based character runs, vaults, jumps, and drop-rolls across a terrain in parkour style in a real-time simulation. Given a single motion capture clip of each of these four skills as input, an offline learning process develops robust feedback control policies for parameterized versions of these skills, as well as transition motions.

and landing. $E(\boldsymbol{M})$ controls desired contacts and penalizes unwanted collisions. It is the most important term for guiding clearing maneuvers to overcome obstacles. For vaulting we need to guide both feet to overpass the obstacle; to guide the hand onto the center of the obstacle for supporting the full body; to control the contact position of the first landing foot; and to control the hand position with respect to CoM to stay close to the reference, because the hand supports the body when both feet are in the air similar to what a stance foot does during normal locomotion. We refer the readers to [26] for the exact definitions of these terms.

The vaulting feedback controller thus learned from a single example motion only works for obstacles that are close in height to that of the example motion. We can, however, further employ the same policy optimization approach to obtain a series of parameterized feedback controllers that work for a larger range of parameters. For example, we can achieve running at various speeds and turning rates, and obstacle clearing maneuvers over barriers of varying heights. These parameterized running and obstacle clearing controllers can be further composed together to achieve parkour-style terrain crossing, as shown in Figure 10 if additional optimizations are employed to adapt the feedback policies during the transition periods between different maneuvers. We refer interested readers to [26] for more details.

3.5 Guided SAMCON

The method for learning feedback policies as described in the last section has several limitations. First, the feedback policies are learned on a large time scale, either for the entire motion or for distinctive phases of motion skills. This re-

quires some trial-and-error to test if a single feedback policy works for the entire motion, or domain knowledge to segment the motion into reasonable phases. This limits the robustness of the learned policies and necessitates special treatments in order to achieve motion transitions. Ultimately we desire a feedback policy for each control fragment, so to react to perturbations and changes robustly in a flexible fashion. The smaller granularity on the scale of $0.1s$ also enables a unified learning process for all motion skills and motion transitions, thus reduces the manual work on motion segmentation and objective function design. However, learning time-indexed policies significantly increase the dimensionality of the parameter space, posing a significant challenge to derivative-free optimization methods such as CMA, even when coupled with the sliding window mechanism and reduced-order feedback.

Inspired by the recent advances on guided policy search [21,22,31], we design an iterative guided learning process, which we refer to as guided SAMCON hereafter, for learning linear feedback policies for each control fragments: the SAMCON method serves as a control oracle that provides high-quality solutions in the form of state-action 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 SAMCON; it provides the control 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.

We construct high-quality open-loop control trajectories from the input motion clips using the improved SAMCON algorithm, and initialize the control fragments with the resulting open-loop controls. Starting from the end state of the last control fragment $\boldsymbol{s}_{k-1}$, each execution of a control fragment $\mathcal{C}_k$ results in a simulation tuple $\tau = (\boldsymbol{s}_{k-1}, \boldsymbol{a}_k, \boldsymbol{s}_k)$. Guided SAMCON uses the current policy of each control fragment $\mathcal{C}_k$ as the distribution to sample from:

$$\begin{aligned} \boldsymbol{a}_k &= \boldsymbol{\pi}(\boldsymbol{s}_{k-1}; \boldsymbol{\theta}_k) \\ &= \boldsymbol{\pi}(\boldsymbol{s}_{k-1}; \boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k) \\ &\sim \mathcal{N}(\boldsymbol{M}_k \boldsymbol{s}_{k-1} + \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k) \end{aligned} \tag{17}$$

$\boldsymbol{\theta}_k = \{\boldsymbol{M}_k, \hat{\boldsymbol{a}}_k, \boldsymbol{\Sigma}_k\}$ as we superimpose Gaussian explorations onto linear deterministic feedback policies. We use a diagonal covariance matrix $\boldsymbol{\Sigma}_k$ with the assumption that each dimension of the action space is independent. This can be viewed as an enhancement of the basic SAMCON (§3.2) that employed a fixed sampling distribution, $\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_0)$, and also of the improved SAMCON (§3.3) 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.

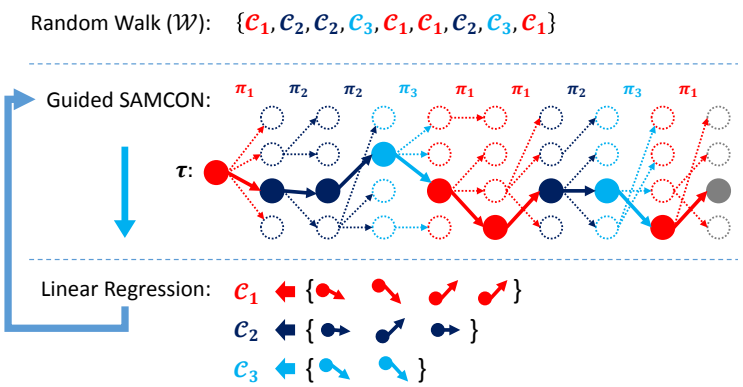


Fig. 11. Schematic illustration of the guided SAMCON process.

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 [46,26]. 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.

We now describe the details of the linear regression for our problem. Then we will derive the iterative use of the linear regression model from an EM-based (expectation maximization) policy search algorithm, which readers can safely choose to skip without affecting the understandability and implementation of the guided SAMCON.

Estimation of Linear Feedback Policy The linear regression problem for control fragments k yields a model to predict $\boldsymbol{a}$ as an affine function of $\boldsymbol{s}$, as per equation 17, where

$$\boldsymbol{M}_k = \left[(S_k^T S_k)^{-1}(S_k^T A_k)\right]^T \tag{18}$$

$$\hat{\boldsymbol{a}}_k = \bar{\boldsymbol{a}}_k - \boldsymbol{M}_k \bar{\boldsymbol{s}}_{k-1} \tag{19}$$

$$\mathrm{diag}(\boldsymbol{\Sigma}_k) = \frac{1}{N_\tau}\,\mathrm{diag}\left[(A_k - S_k \boldsymbol{M}_k^T)^T (A_k - S_k \boldsymbol{M}_k^T)\right] \tag{20}$$

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

$$S_k = \left[\boldsymbol{s}_{k-1}^1 - \bar{\boldsymbol{s}}_{k-1}, \ldots, \boldsymbol{s}_{k-1}^{N_k} - \bar{\boldsymbol{s}}_{k-1}\right]^T \tag{21}$$

$$A_k = \left[\boldsymbol{a}_k^1 - \bar{\boldsymbol{a}}_k, \ldots, \boldsymbol{a}_k^{N_k} - \bar{\boldsymbol{a}}_k\right]^T \tag{22}$$

The feedback policies are initialized to zero, i.e. $\boldsymbol{M} = \mathbf{0}, \hat{\boldsymbol{a}} = \mathbf{0}$. To prevent the regression from being underdetermined, the control fragment sequence $\mathcal{W}$ has to be long enough so that $N_k \geqslant 200$ for all control fragments. This is easy to achieve for cyclic motions, i.e., we simply repeat the cycle 200 times. For non-cyclic motion skills and motion transitions, we incorporate *control graphs*, a graph structure that will be detailed in Section 3.6, to generate a long random walk $\mathcal{W}$ on the graph such that each control fragment appears at least 200 times in $\mathcal{W}$. All the sample tuples $\{\tau_k^j\}$ for all the control fragments $\{\mathcal{C}_k\}$ can be collected simultaneously by generating a successful execution of $\mathcal{W}$ using the guided SAMCON, represented by $\boldsymbol{\tau} = \{\tau_{k_1}, \tau_{k_2}, \dots\}$, as shown in Figure 11. Then we can extract the simulation tuples for each individual control fragment to update the feedback parameters as in Equations 18∼20.

We further regularize the Frobenius norm of the feedback gain matrix $\boldsymbol{M}_k$, so that

$$\boldsymbol{M}_k = \left[(S_k^T S_k + \lambda I)^{-1}(S_k^T A_k)\right]^T \tag{23}$$

is used instead of Equation 18. We use $\lambda = 10^{-6}$ in all our experiments. The feedback gains together with the observed prediction variances as captured by $\boldsymbol{\Sigma_k}$ provide a stochastic version of the control policy that will be used to guide the sampling used by the basic SAMCON that serves as our control oracle:

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

Guided Learning as EM-based Policy Search Given a reward function $R(\tau)$ that measures the goodness of a simulation tuple $\tau = (\boldsymbol{s}_{k-1}, \boldsymbol{a}_k, \boldsymbol{s}_k)$, resulted from the execution of a control fragment $\mathcal{C}_k$, policy search seeks for the optimal policy that maximizes the expected return

$$J(\boldsymbol{\theta}) = \int_\tau P(\tau;\boldsymbol{\theta})R(\tau) \tag{24}$$

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

$$P(\tau;\boldsymbol{\theta}) = P(\boldsymbol{s}_k|\boldsymbol{s}_{k-1},\boldsymbol{a}_k)\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})P(\boldsymbol{s}_{k-1}) \tag{25}$$

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.

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. It is shown in [35] that the iterative update procedure of an EM algorithm applied to episodic policy search for a linear policy 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:

$$\log \frac{J(\boldsymbol{\theta})}{J(\boldsymbol{\theta}_0)} = \log \int_\tau P(\tau;\boldsymbol{\theta}) R(\tau)/J(\boldsymbol{\theta}_0) \tag{26}$$

$$\geqslant \frac{1}{J(\boldsymbol{\theta}_0)} \int_\tau P(\tau;\boldsymbol{\theta}_0) R(\tau) \log \frac{P(\tau;\boldsymbol{\theta})}{P(\tau;\boldsymbol{\theta}_0)} \tag{27}$$

$$\propto \int_\tau P(\tau;\boldsymbol{\theta}_0) R(\tau) \log \frac{\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta})}{\boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}_0)} \tag{28}$$

$$= L(\boldsymbol{\theta};\boldsymbol{\theta}_0) + C \tag{29}$$

where Equation 27 applies Jensen's inequality to the concave logarithm function, C is a constant independent of $\boldsymbol{\theta}$ and

$$L(\boldsymbol{\theta};\boldsymbol{\theta}_0) := \int_\tau P(\tau;\boldsymbol{\theta}_0) R(\tau) \log \boldsymbol{\pi}(\boldsymbol{a}_k|\boldsymbol{s}_{k-1};\boldsymbol{\theta}) \tag{30}$$

Note that the optimal $\boldsymbol{\theta}$ must satisfies $J(\boldsymbol{\theta}) \geqslant J(\boldsymbol{\theta}_0)$ because Equation 27 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 $\{\tau_k^j\}$ sampled according to the current policy $\boldsymbol{\pi}_k(\boldsymbol{a}_k|\boldsymbol{s}_{k-1},\boldsymbol{\theta}_0)$ as:

$$L(\boldsymbol{\theta};\boldsymbol{\theta}_0) \approx \frac{1}{N_k} \sum_{j=1}^{N_k} R(\tau^j) \log \boldsymbol{\pi}_k(\boldsymbol{a}_k^j|\boldsymbol{s}_{k-1}^j;\boldsymbol{\theta}) \tag{31}$$

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

$$\begin{aligned} \mathbf{0} &= \frac{\partial}{\partial\boldsymbol{\theta}} L(\boldsymbol{\theta};\boldsymbol{\theta}_0) \\ &\propto \sum_{j=1}^{N_k} R(\tau^j) \frac{\partial}{\partial\boldsymbol{\theta}} \log \boldsymbol{\pi}_k(\boldsymbol{a}_k^j|\boldsymbol{s}_{k-1}^j;\boldsymbol{\theta}) \end{aligned} \tag{32}$$

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.

We assign a constant reward to all simulation tuples, which implies a special reward function in the form of

$$R(\tau) = \begin{cases} 1 & \text{tuple } \tau \text{ is good enough in the long run so} \\ & \text{that the random walk } \mathcal{W} \text{ can succeed.} \\ 0 & \text{otherwise.} \end{cases} \tag{33}$$

Solving Equation 32 against this reward function and the Gaussian Explorations of Equation 17 leads to the linear regression that we described in the last section.

Results A variety of individual cyclic skills have been tested to fully evaluate the capability of the guided SAMCON learning framework, including basic

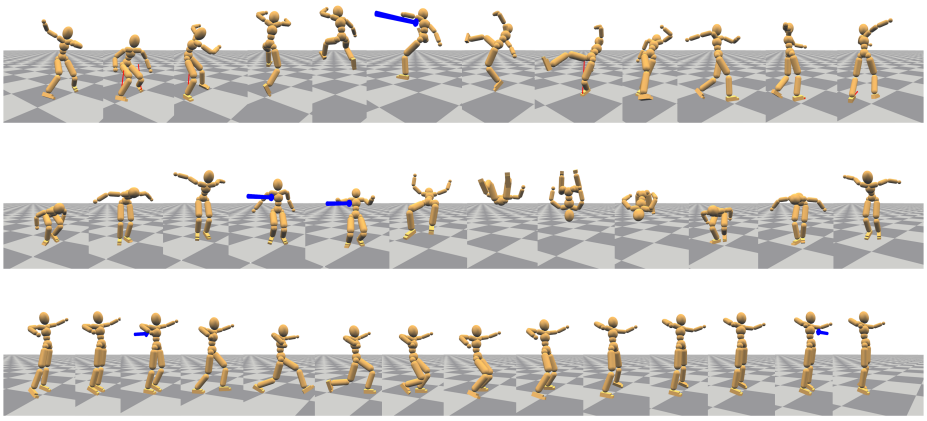


Fig. 12. Simulations of learned skills under external pushes. Top: kick. Middle: backflip. Bottom: waltz.

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 12 demonstrate the executions of several learned skills.

The learned skills are robust enough to enable repeated executions even under external perturbations. In Figure 12 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. Generally speaking, faster motions such as running take fewer learning iterations to achieve robust cyclic motions, as well as tolerating larger perturbations. In contrast, slow motions such as the balancing phase of the backflip are more sensitive to perturbations. This may be explained in part by the additional balance opportunities afforded by each new contact phase. We also note that we run the learning process for a maximum of twenty iterations, given that this allows robust feedback policies to be found for all motions tested. However, none of the motion skills can acquire robust feedback policies with only one iteration. This suggests a linear regression with the basic SAMCON, equivalent to the first iteration of Guided SAMCON with a zero policy, is not enough to achieve robust skills.

The robustness of the guided SAMCON 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 controls initialized and refined from the results built for our default character. Figure 13 shows several examples where we retarget the cyclic kick and the

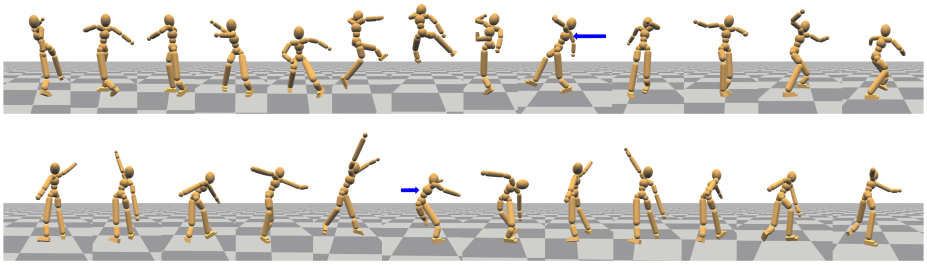


Fig. 13. Retargeting kick (top) and dance-spin (bottom) to characters with modified body ratio.

dance spin to characters with modified body segment lengths. The retargeted controllers are robust to external perturbations as before.

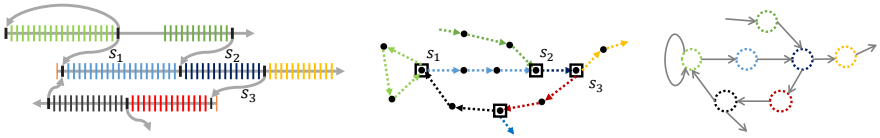
All the test skills can be learned with the standard settings as described thus far, while special treatment is applied for walking and running in order to achieve symmetric gaits. Specifically, we pick one stride (half step) from the example clip and concatenate its mirror stride to generate a symmetric 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. Enforcing symmetry is not necessary for finding robust control policies. However, the resulting asymmetric gaits often make the character turn slightly to one direction. 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 develop parameterized turns for these locomotion skills, as we can record the corresponding actions under external pushes and add them to the action vectors in order to make the character perform shallow turns. We use this simple parameterization method to implement basic steering behaviors in our demonstrations. For rapid turns we still need to use controllers learned from relevant motion capture examples.

We further employ contact-aligned phase-resetting for walking and running controllers, which improves their robustness to large perturbations. In contrast, we have noted that contact-aligned phase-resets are neither necessary nor helpful for learning controllers for complex skills such as kicks and backflips. Motions that involve long airborne phases might be considered sensitive to the contact events at take-off and landing. However, our pipeline is capable of finding successful control policies without contact alignment at the control fragment level or any other special treatment of these critical instants. In practice, the learned policies are robust to the simulated contact events occuring in fragments that immediately precede or follow the nominal fragment for that event.

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

Skills	T_{cycle} (s)	t_{learning} (min)
Catwalk	0.7	40.3
StridingRun	0.45	21.1
Waltz	5.0	314
Kick	1.6	93.8
DanceSpin	1.6	102
Backflip	2.5	153

Table 2. 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.



(a) a motion graph (b) a control graph (c)

Fig. 14. 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 represents a chain of control fragments, or rather, a controller.

to the length of the clip, the number of samples N_s, and inversely with the number of cores available. Table 2 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.

3.6 Control Graphs

The guided SAMCON as described in the last section can be directly applied to cyclic motions. For non-cyclic motions and motion transitions, we need a mechanism to apply the basic SAMCON to multiple instances of their control fragments, in order to collect enough samples for the policy updates in the form of linear regressions. Inspired by the successful motion graphs [17] for kinematic motion synthesis, we organize the control fragments into a graph as shown in Figure 14(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 14(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 14(c) we then illustrate the corresponding connectivity between controllers. Here, an arrow indicates that the controller associated with the *tail* ends in

Algorithm 2 Guided Learning of Control Graphs

Require: example motion clips of skills
Ensure: 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 $\mathcal{W} = \{\mathcal{C}_{k_1}, \ldots, \mathcal{C}_{k_N}\}$
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: $\{\tau_k^i\} \leftarrow$ 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**

a state that is close 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 14(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 again initialize the open-loop controls for the control fragments using the improved SAMCON algorithm, and initialize the feedback policies $\boldsymbol{\pi}$ with zero.

Algorithm 2 summarizes the 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 chapter, 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). Building high-quality motion graphs can be a non-trivial task, even with the help of automated techniques such as the one proposed by [17]. Manual 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 chapter, we man-

ually specify the connectivity of the motion graphs for our control graphs by selecting visually similar poses as transition points. We then apply kinematic blending to a few frames of the motion clips near the transition points. Our learning procedure is generally robust to kinematic flaws due to such blending or noise, and is able to generate high-quality simulated motions. Occasionally, selected transitions based on pose similarity alone may result in poor results, such as those between slow motions and fast motions where velocities should also be taken into account. In such cases, we perform a search for the best transition point around the originally selected point, using a $0.05s$ sampling interval. We can find successful transitions within ten samples in all such cases.

Refining Open-loop Controls: The initial open-loop controls of each control fragment are computed by the improved SAMCON from the individual motion example clips, which are not physically plausible when directly connected together by a random walk on 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 basic SAMCON again 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 improves the possibility of finding a robust feedback policy that can deal with all possible transitions involved, similar to its noise reduction effect for the improved SAMCON.

Learning Feedback Policies: In line 5, the feedback policies are initialized, as well as the default exploration covariances. We find that $\sigma_0 = 5^\circ$ 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 random walk sequence, especially in the first iteration when the initial zero-valued policy is applied, where Guided SAMCON degenerates into the basic SAMCON. 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.

Progressive Learning: Learning the controllers for all the motion skills of a control graph simultaneously can be inefficient, because the learning for different controllers converges at different speeds, 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 control graphs progressively. Figure 15 illustrates two prototype control graphs and the example learning orders

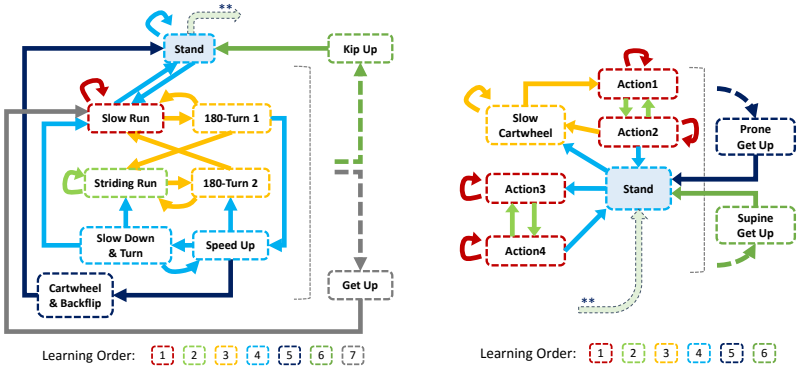


Fig. 15. Two porotype control graphs progressively learned in the order marked in different colors. Only major controllers are shown in the graph for clarity. The rising skills indicated by dashed arrows will be triggered automatically once the character falls. Left: Locomotion and gymnastics graph. Right: Bollywood dancing graph. Action1–arm hip shake; Action2–chest pump+swag throw; Action3–pick and throw; Action4–hand circular pump.

we use. Specifically, we start from learning controllers for a few cyclic skills. Non-cyclic skills are then gradually incorporated into the latest subgraph by re-running the whole learning pipeline. This progressive process skips the learned skills from guided SAMCON by locking the learned policies instead of generating additional exploratory samples for further learning. Our experiments show that the learned feedback policies from the smaller graph are robust enough to deal with new transitions in the enlarged graph. However, it is possible that the newly added transitions are not compatible with existing control policies. We suggest simply unlock the learned controllers connected with the new skill for further adapting their parameters. 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.

Results Figure 15 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. We learn the control graphs progressively in the order illustrated in Figure 15. We always start by learning cyclic controllers, using the process just described. Non-cyclic skills are then gradually incorporated into the latest subgraph by rerunning the whole learning pipeline.

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

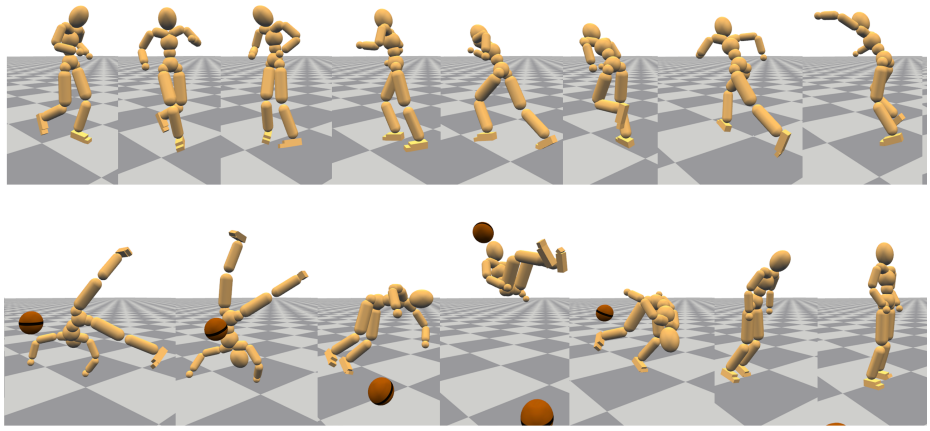


Fig. 16. Random walk on the control graph with external perturbations.

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.

The two control graphs together contain a total of 40 seconds of reference motion. The aggregate construction time for our motion graphs, beginning from the reference motions, is approximately two full days on a 20-core computer cluster, including both user-in-the-loop work and offline computation. This begins with the computation of the open-loop trajectory from the original reference motion using SAMCON. The compute time for this depends on the quality of the reference motion, varying from as few as ten minutes for many motions, i.e., runs, bollywood dances, get-up, kip-up, to one hour or more if the improved SAMCON algorithm is required, as was the case for flips and gymnastic motions. The guided learning procedure usually takes up to one hour to learn a one-second long skill. In practice, we often stop well before the maximum 20 EM-iterations, resulting in compute times under one hour for a one-second cyclic skill. In aggregate, we find that it takes between 1–2 hours per second of reference motion, which includes both the required manual steps and the computational time on a 20-core cluster. In practice, we can get through all the manual steps in the daytime and run the learning procedures overnight with batch-mode settings.

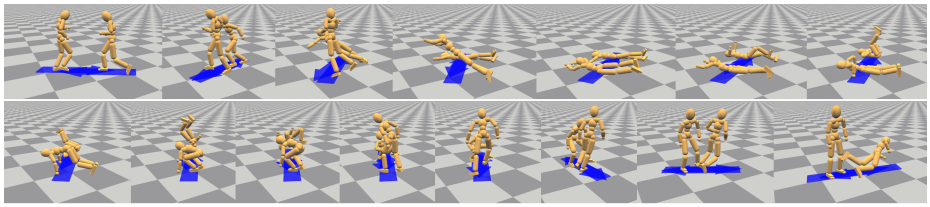


Fig. 17. Two simulated characters try to run into each other. Both of them are controlled by the same control graph.

We demonstrate several applications of the prototype control graphs. 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 for steering, we can easily achieve interactive navigation in a scene. The characters can also robustly perform the desired motion skills in the presence of moderate external perturbations such as pushes on the trunk and ball-impacts as shown in Figure 16. The character will fall if it is disturbed too much, which automatically activates the rising controllers that will return the character to performing desired motions designated by the high-level planner.

Figure 17 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. More results and video demonstrations can be found in [24].

3.7 Discussion

We have presented two sampling-based methods, the basic SAMCON and the improved SAMCON, for constructing open-loop controls for individual motion skills from example motion clips. We have also described two learning methods, namely reduced-order feedback learning and guided SAMCON, for learning closed-loop feedback policies from the open-loop controls reconstructed by the SAMCON algorithms. In particular, the guided SAMCON framework integrates all the successful ingredients of sampling-based control reconstruction and policy search. Therefore using guided SAMCON, we can develop control graphs 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, which opens the door to the possible use of physics-based character motions in sports scenarios. In all our experiments, guided SAMCON always succeeds if the open-loop controls can be constructed. A failure to re-

construct the open-loop controls is usually due to poor quality of the reference motion or improper selection of transition points.

Compared with robust feedback policies that are specially tailored to locomotion, such as SIMBICON-type controllers [51,20], our framework works for a much richer class of motions including many that have rapid contact changes. We do note that our learned walking controller is less robust than the walking gait used to measure the robustness in the original SIMBICON paper [51]. This is similar to earlier results by [46], who also find that their natural walks are less robust to external force perturbations than the original SIMBICON walking gait. The robustness of a walking gait is thus related to the particular walking style as well as the specific feedback system. In comparison to model-based approaches such as quadratic programming [29,5] and differential dynamic programming [32], our method does not require exact knowledge of the dynamics models or carefully-tuned optimization objectives and constraints. In addition, we incorporate motion transitions in the same framework, thereby enabling multi-skilled 3D avatars that are capable of real-time interaction with the environment and with each other. To the best of our knowledge, the guided SAMCON framework is one of the most general approaches for learning controllers for physics-based characters through simulations.

Advantages of Sampling-based Methods When used for optimization, sampling-based approaches do not demand derivative computation, in contrast to gradient-based techniques. This is useful when derivatives are difficult or impossible to compute. Many physics-based animation systems are developed on top of third-party simulators, which can preclude the computation of analytic derivatives. It is also well-known that derivatives are difficult to compute for tasks with abundant transient contacts, such as a roll-and-get-up motion. These contacts pose a serious challenge to inverse dynamics algorithms and gradient-based optimizations. For situations where gradient computations are plausible, gradient-based techniques are nevertheless prone to local minima for highly nonlinear problems in high dimensions. Derivative-free sampling techniques are not immune to local minima, but they can nevertheless often escape a local minimum.

When applied to creating motions, a side benefit of sampling-based methods is that the stochastic nature of the solution will naturally exhibit a degree of motion variation. Motion synthesis is often cast as an optimization problem, based on the assumption that desired motions are optimal in some sense. However, this ignores the natural variations that are evident in human motion. Sampling-based methods can also work to achieve a given goal in the absence of a reference trajectory, although having one greatly prunes the search space and accelerates the construction. This allows us to generate control sequences for non goal-oriented tasks, such as idling, where a desired trajectory is hard to specify or capture [27]. Sampling schemes can potentially discover new strategies, given enough computational resources.

Sampling-based techniques are also easy to parallelize, which is of importance in the continuing era of multi-core computers and compute clusters. We show that control for individual complex tasks can be reconstructed within minutes, and control for a reasonably-sized graph can be reconstructed within a day or two, on small-scale clusters.

Automation 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.

After the initialization procedures, the guided SAMCON 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.

4 Future Work

There remain many open problems for further investigation in the near future. We are particularly interested in further pushing forward the robustness and generalization ability of controllers through simulation; and applying the strategies developed in simulation to real robots. We conjecture that better control representations need to be devised, and more powerful learning frameworks need to be explored.

Control Representation In the SAMCON family algorithms, the controllers, as defined by sequences of control fragments, implicitly define time-indexed piece-wise policies. The default control fragment duration, $0.1s$, represents a compromise between an excessively long duration, which eventually leads to a feedback structure that is insufficiently flexible to provide robust control, and an excessively short duration, which increases the compute time and may be prone to over-fitting or local minima. In practice, we find that control fragment durations in the range of $0.05s-0.2s$ also provide robust solutions. The advantage of short-duration control representation as compared to longer-duration controls is that manual segmentation into motion phases and optimization design for each phase can be avoided; using a fixed linear policy across all phases of a motion is generally insufficient for complex motions and motion transitions. Our

regression-based learning can efficiently learn the large number of linear feedback parameters that result from the use of a unique linear model dedicated to each control fragment.

This scheme mitigates many difficulties in learning the feedback policies, but makes the learned policies less flexible. One disadvantage of such control is the dependency of the control on the time index of the reference motions. In addition, our method assumes there is no need for multi-modal responses, such as would arise in the case of a policy that should steer either left or right to avoid a collision, but should not steer straight. As future work, we wish to develop state-indexed feedback policies by using data obtained from the time-indexed policies to learn richer state-based policy representations, such as Gaussian mixture models or neural networks [21,31,41]. Moreover, our controllers require reference trajectories and/or open-loop feedforward controls for tracking, even though no dynamics models are necessary. To achieve a more powerful and robust control system, we envision a hybrid approach that combines the strengths of model-free and model-based control methods.

Learning Framework We follow the recent successful guided policy search framework, 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. Our learning pipeline is unique in its use of: (1) the use of an implicit-dynamics, sampling-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 multi-skilled characters. Therefore we are able to generate controllers with state-of-the-art skill repertoires. We wish to continue pushing the current learning methods for better performance. However, we note that the current framework cannot be directly applied to motion skills that involve non-trivial human-object interactions such as dribbling a soccer ball or non-trivial interactions between multiple characters such as social dancing or wrestling.

We wish to experiment with deep learning methods, which have been integral to recent breakthroughs in speech recognition and computer vision. Several recent studies have shown promising results by applying deep learning for motion control problems[23,39]. However, current results still fall well short of being capable of skilled control of human-like motions. Our sampling-based methods are suitable for generating large datasets, and our learned linear policies may provide good bootstrapping strategies for training deep neural networks. Among other benefits, the deep networks may provide a way to learn the state and action representations that are best suited for motor control, and to enable new motor skills that the current learning framework may not be capable of.

Generalization In computer character animation, one of the major motivations for physics-based motion synthesis techniques is to generalize motion capture data [52,49]. We have demonstrated several forms of motion generalization

within the same framework, but also foresee a number of further developments. *Motion cleanup:* Contact-rich motions are hard to capture and clean up. Some of the input trajectories we use have serious contact flaws, like ground penetrations and contact sliding. Our method currently can correct penetrations but not sliding. We can experiment with adding another term into the cost function to penalize sliding and other artifacts in the input that we wish to get rid of. *Motion variations:* We focus on small variations that can be treated as noise, i.e., the same person in the same physical and mental conditions. For motion variations caused by other reasons, such as mood changes or physical injuries, new mechanisms have to be devised. *Motion parameterization:* To achieve a rich motion repertoire, parameterized feedback controllers that work for a large range of task-related parameters or environmental parameters are desirable. We have been successful in parameterizing skill-level or phase-level reduced-order feedback controllers, but we have yet to experiment with parameterized controllers based on control fragments. *Motion transformation:* Because of the tracking nature of the trajectory-based sampling, our motion transformations cannot deviate too far away from the input trajectory. To achieve even larger transformations, non-tracking control schemes such as model-based methods that do not hold on to the reference trajectory all the time are necessary. Manual editing of the input trajectories can also help shape the synthesized motions. *Motion retargeting:* We have retargeted several motions to an Asimo-like robot model, which differs significantly from human models. If the models were to differ more, at some point the retargeting would simply fail. Continuation methods may be helpful for more aggressive retargeting [50].

Transfer Control to Robots A number of challenges remain to be resolved in order to effectively apply the control strategies developed in this chapter to real robots. Several sources of mismatch between control in simulation and for real robots have been presented in the introduction. Given uncertainties in estimating the kinematic and dynamic parameters of the system, it needs to be ensured that the control policies are robust with respect to the resulting errors. Approaches to this include ensemble optimization methods [30] and iterative updates of the parameters and policies [13]. Uncertainties may also arise with respect to sensor readings, including the readings being asynchronous and delayed. This requires another dimension of robustness for the control policy. Because the PD-controllers used in our simulations have gains that are likely to be quite different from those seen on low-cost robotic hardware, it makes more sense to view our PD controllers as a computational step to compute the desired joint torques for torque-controlled humanoid hardware. Lastly, as others have noted in the past, e.g., [37], human motions may need to be explicitly retargeted and retimed in order to be within the capabilities of a given robot. This same problem exists when athletes of different dimensions and proportions that attempt to perform a given motion. As in the case of athletes attempting to reproduce a given motion, the solution is likely to require imitating an overall motion style or motion goals instead of more directly trying to track the motion in joint space.

Acknowledgements We sincerely thank all our collaborators for their contributions of the work described in this chapter, especially Ding Kai, Weiwei Xu, Tianjia Shao, and Baining Guo. This work was funded in part by NSERC Discovery Grant RGPIN-2015-04843.

References

1. Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics*, 19(8):1405–1414, 2013.
2. Stephen Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *SIGGRAPH'00*, pages 219–228, 2000.
3. Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Trans. Graph.*, 28(5):170:1–170:9, 2009.
4. Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Trans. Graph.*, 29(4):130:1–130:9, 2010.
5. Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Trans. Graph.*, 29(4):131:1–131:10, 2010.
6. Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. Learning reduced-order feedback policies for motion skills. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA'15, pages 83–92. ACM, 2015. `doi: 10.1145/2786784.2786802`.
7. Arnaud Doucet, Nando de Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo Methods in Practice*. Springer, 2001.
8. Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In *Handbook of Nonlinear Filtering*. Oxford, UK: Oxford University Press, 2011.
9. Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *ICRA'15*, pages 4397–4404, 2015.
10. Stevie Giovanni and KangKang Yin. Locotest: Deploying and evaluating physics-based locomotion on multiple simulation platforms. *Lecture Notes in Computer Science*, 7060:227–241, 2011.
11. Ambarish Goswami, Seung kook Yun, Umashankar Nagarajan, Sung-Hee Lee, KangKang Yin, and Shivaram Kalyanakrishnan. Direction-changing fall control of humanoid robots: Theory and experiments. *Autonomous Robots*, 36(3):199–223, 2014.
12. Sehoon Ha and Katsu Yamane. Reducing hardware experiments for model learning and policy optimization. In *ICRA'15*, pages 2620–2626, 2015.
13. Sehoon Ha and Katsu Yamane. Reducing hardware experiments for model learning and policy optimization. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2620–2626. IEEE, 2015.
14. Nikolaus Hansen. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer Berlin Heidelberg, 2006.

15. Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *SIGGRAPH'95*, pages 71–78, 1995.
16. Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *Int. J. Comput. Vision*, 29(1):5–28, 1998.
17. Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH'02*, pages 473–482, New York, NY, USA, 2002. ACM.
18. James J. Kuffner, Jr., Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots. *Auton. Robots*, 12(1):105–118, 2002.
19. S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.
20. Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. *ACM Trans. Graph.*, 29(4):129:1–129:8, 2010.
21. Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML'13*, 2013.
22. Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *ICML'14*, 2014.
23. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.
24. Libin Liu, Michiel van de Panne, and KangKang Yin. Guided learning of control graphs for physics-based characters. *ACM Trans. Graph.*, 35(3):Article 29, 2016. doi:10.1145/2893476.
25. Libin Liu, KangKang Yin, and Baining Guo. Improving sampling-based motion control. *Computer Graphics Forum*, 34(2):415–423, 2015. doi:10.1111/cgf.12571.
26. Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph.*, 31(6):Article 154, 2012. doi:10.1145/2366145.2366173.
27. Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. *ACM Trans. Graph.*, 29(4):Article 128, 2010. doi:10.1145/1778765.1778865.
28. Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. Simulation and control of skeleton-driven soft body characters. *ACM Trans. Graph.*, 32(6):Article 215, 2013. doi:10.1145/2508363.2508427.
29. Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Trans. Graph.*, 28(3):Article 80, 2009.
30. Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *IROS*, pages 5307–5314. IEEE, 2015.
31. Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, Berkeley, USA, July 2014.
32. Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.*, 28(3):Article 81, 2009.
33. Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Dynamic terrain traversal skills using reinforcement learning. *ACM Trans. Graph.*, 34(4):80:1–80:11, 2015.
34. Jan Peters and Jens Kober. Using reward-weighted imitation for robot reinforcement learning. In *Adaptive Dynamic Programming and Reinforcement Learning*, pages 226–232, March 2009.
35. Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *ICML'07*, pages 745–750, 2007.

36. Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *NEURAL NETWORKS*, 21(4):682–697, MAY 2008.
37. Nancy S Pollard, Jessica K Hodgins, Marcia J Riley, and Christopher G Atkeson. Adapting human motion for the control of a humanoid robot. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1390–1397. IEEE, 2002.
38. Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *SIGGRAPH'00*, pages 209–217, 2000.
39. Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *ICRA'15*, pages 3223–3230, 2015.
40. K. Sims. Evolving virtual creatures. In *SIGGRAPH'94*, pages 15–22, 1994.
41. Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. Learning bicycle stunts. *ACM Trans. Graph.*, 33(4):50:1–50:12, 2014.
42. Jie Tan, C. Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Comput. Graph. Appl.*, 31(4):34–44, 2011.
43. Konstantinos I. Tsianos, Ioan Alexandru Sucan, and Lydia E. Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1:2–11, August 2007.
44. Christopher D. Twigg and Doug L. James. Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph.*, 26(3), 2007.
45. Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Trans. Graph.*, 28(3):Article 60, 2009.
46. Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers. *ACM Trans. Graph.*, 28(5):Article 168, 2009.
47. Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH'88*, pages 159–168, 1988.
48. Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph.*, 23(3):532–539, 2004.
49. KangKang Yin, Michael B. Cline, and Dinesh K. Pai. Motion perturbation based on simple neuromotor control models. In *PG'03 (Pacific Conference on Computer Graphics and Applications)*, pages 445–449, 2003.
50. KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Continuation methods for adapting simulated skills. *ACM Trans. Graph.*, 27(3):Article 81, 2008.
51. KangKang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.
52. Victor B. Zordan and Jessica. K. Hodgins. Motion capture-driven simulations that hit and react. In *SCA (ACM SIGGRAPH/Eurographics Symposium on Computer Animation)*, pages 89–96, July 2002.