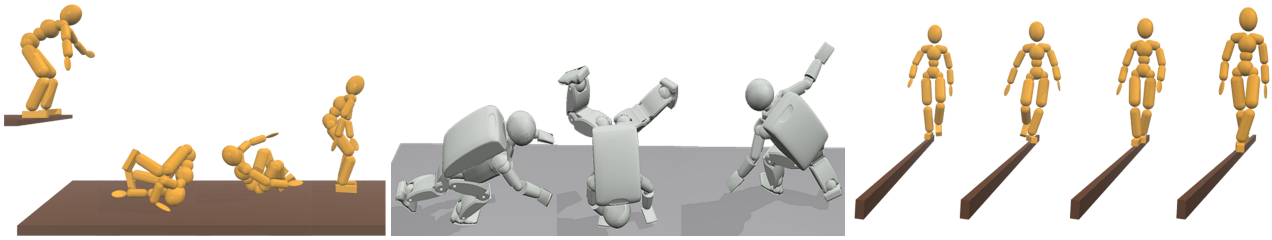
Sampling-based Contact-rich Motion Control

Libin Liu* KangKang Yin† Michiel van de Panne‡ Tianjia Shao* Weiwei Xu†

* Tsinghua University † Microsoft Research Asia ‡ University of British Columbia



(a) A forward roll transformed to a dive roll. (b) A cartwheel retargeted to an Asimo-like robot. (c) A walk transformed onto a balance beam.

Figure 1: *Physically based motion transformation and retargeting.*

Abstract

Human motions are the product of internal and external forces, but these forces are very difficult to measure in a general setting. Given a motion capture trajectory, we propose a method to reconstruct its open-loop control and the implicit contact forces. The method employs a strategy based on randomized sampling of the control within user-specified bounds, coupled with forward dynamics simulation. Sampling-based techniques are well suited to this task because of their lack of dependence on derivatives, which are difficult to estimate in contact-rich scenarios. They are also easy to parallelize, which we exploit in our implementation on a compute cluster. We demonstrate reconstruction of a diverse set of captured motions, including walking, running, and contact rich tasks such as rolls and kip-up jumps. We further show how the method can be applied to physically based motion transformation and retargeting, physically plausible motion variations, and reference-trajectory-free idling motions. Alongside the successes, we point out a number of limitations and directions for future work.

1 Introduction

Motion capture has been widely used for generating high quality character animations. However, it remains expensive because of the equipment and space required, as well as frequently requiring tedious manual post-processing of the data. Contact-rich motions are particularly difficult to capture and edit afterwards. Self-contacts or occlusions caused by contacts with the environment pose serious problems for optical motion tracking systems. Post-processing such motions is challenging even for professional animators because of the simultaneous existence of many contact constraints. Moreover, manually fixing violated kinematic constraints can easily destroy the physical realism of the original motion. Delicate spatiotemporal relationships are embedded in the motion dynamics among all the degrees of freedom (DoFs) for goal-oriented tasks such as rolling.

Reusing and generalizing captured motions or keyframed motions is the focus of much animation research today [Kovar et al. 2002; Kim et al. 2009]. Of all such efforts, physics-based animation techniques promise broader generalizations while preserving physical realism [Sok et al. 2007; Yin et al. 2007; Muico et al. 2009; Macchietto et al. 2009]. Spacetime trajectory optimizations incorporate physical constraints while searching for or transforming trajectories [Witkin and Kass 1988; Liu et al. 2006]. Optimization while tracking reference trajectories during simulations can generate high quality motions [Muico et al. 2009; Macchietto et al. 2009]. However, the contact states of the generated motions cannot deviate significantly from the reference trajectory. Proportional derivative (PD) control coupled with realtime foot placement feedback can provide robust locomotion, e.g., [Yin et al. 2007], but it is not clear if it generalizes to other tasks. Our goal is to develop a general method that can compute controls for multiple types of tasks, some of which do not have well-defined, repeatable contact states. A caveat will be that our results are computed offline.

We propose to reconstruct the control underlying a motion by casting it as a search problem that seeks to follow a given reference trajectory. Randomized sampling supports this search by creating a diverse set of motion sequences from which a 'best tracking' path can then be selected. Our use of sampling is inspired by past successes of sampling-based strategies in robot motion planning [Tsianos et al. 2007], passive animation [Chenney and Forsyth 2000; Twigg and James 2007] and active animation [Sims 1994; Wang et al. 2009].

When used for optimization, sampling-based approaches do not demand derivative computation, in contrast to gradient-based techniques. This is useful when derivatives are hard 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

*e-mail:{llb05,shaotj08}@mails.tsinghua.edu.cn
†e-mail:{kkyin,wwxu}@microsoft.com
‡e-mail:van@cs.ubc.ca

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. Sampling schemes can potentially discover new strategies, given enough computational resources.

Sampling-based techniques are also easy to parallelize, which is of importance as multi-core computers and compute clusters become ever more commonplace. We show that control for complex tasks can be reconstructed within minutes on small-scale clusters.

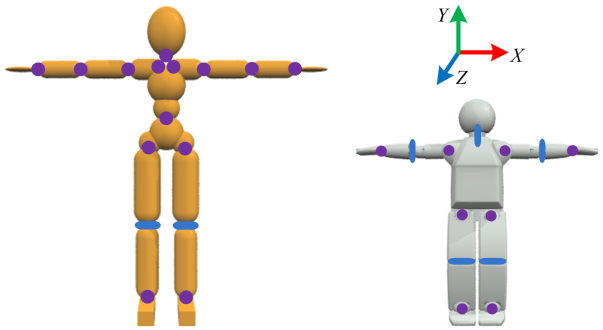
2 Related Work

Motor Control: There are several concepts from the field of motor control that are related to our work: contact dynamics, feedforward control, and feedback control. Motion and interaction with the environment are fundamentally intertwined. Contact dynamics, or the Ground Reaction Force (GRF), is measured and studied intensively in medicine and sports. These studies focus on balance and locomotion, and analysis rather than synthesis. An interesting recent method estimates joint torques and a parametric contact model from motion [Brubaker et al. 2009]. Internal models represent one of the most successful concepts established in neuroscience in recent years [Kawato 1999; Jordan and Wolpert 1999], and they suggest decomposing motor control into a feedforward component and a feedback component.

Motion Planning: Motion Planning is a well-studied problem in Robotics. Randomized sampling algorithms for path planning, such as PRMs (Probabilistic Roadmap Methods) [Kavraki et al. 1996] and RRTs (Rapidly-exploring Random Trees) [LaValle and Kuffner 2000] provide significant benefits in speed and robustness over conventional deterministic planning algorithms. The computer animation community has adopted these ideas for locomotion planning and manipulation planning [Choi et al. 2003; Yamane et al. 2004]. Trajectory planning, also called kinodynamic planning, is a more difficult task than path planning [LaValle 2006]. A trajectory is a path with a time constraint. Thus while path planning only needs to consider kinematic constraints, trajectory planning has to take dynamic constraints into account. In this paper, we reconstruct controls to produce plausible trajectories.

Sampling-based Passive Animation: Sampling physically plausible simulations to satisfy user constraints is demonstrated to be effective for passive rigid body systems [Chenney and Forsyth 2000; Twigg and James 2007]. Sampling in a precomputation stage helps achieve real-time control of deformable objects and fluids at runtime [James and Fatahalian 2003; Barbič and Popović 2008].

Sampling-based Character Control: More relevant to our work is the use of local stochastic search and genetic algorithms with built-in randomness that can develop interesting behaviors or morphologies for virtual creatures [Van de Panne and Fiume 1993; Ngo and Marks 1993; Sims 1994]. Stochastic optimization has also been explored for constructing and adapting controllers for bipedal locomotion [Hodgins and Pollard 1997; Sok et al. 2007; Yin et al. 2008]. More recently, sampling-based optimization methods, such as the covariance matrix adaptation strategy, have been shown to be effective in optimizing walking controllers [Wang et al. 2009] and generating optimal gaits and morphologies for animal locomotion when combined with traditional derivative-based continuous optimization [Wampler and Popović 2009].



(a) (b)

Figure 2: *Collision geometries and DoFs of our character model (a) and robot model (b). Purple dots denote 3-DOF ball joints, and blue spindles denote 1-DOF hinge joints . There are 50 DoFs in total for the character model, and 35 DoFs in total for the robot model, including the global root position and orientation.*

The work of [Sharon and van de Panne 2005; Sok et al. 2007] are the closest in spirit to our own. Sharon and van de Panne [2005] apply a deterministic coordinate-descent method to optimize the control as best as possible to match target walking motions. The optimization proceeds using multiple episodes of increasing simulation duration to deal with overly large search spaces and undesired local minima. Sok et al. [2007] demonstrates control reconstruction for 2D locomotion tasks. Their method randomly chooses initial configurations and uses a downhill simplex method to find local minima. However, the relative benefits of the random initialization and the local optimization are difficult to quantify, and local optimization procedures have a sequential nature that makes them hard to parallelize. Our algorithm demonstrates successful reconstruction for challenging 3D contact-rich motions by focussing its resources fully on stochastic sampling.

3 Sampling-based Control Construction

We now detail our sampling-based method for reconstruction of a motion capture trajectory. We begin with a description of the motion data and our character models.

3.1 Motion Data and Character Models

We use motion capture data from the CMU motion capture database, data of published works, data captured by ourselves, and example trials that came with our motion capture software. These motions demonstrate various tasks and were captured from different subjects by different groups, with different capture process and data postprocessing, and contain varying degrees of noise. A control reconstruction method thus needs to be flexible and robust to handle them all.

Our character model, shown in Figure 2(a), is $1.7m$ tall and weighs $62.5kg$. Its detailed kinematic and dynamic parameters can be found in Table 1 and Table 2. To incorporate motion data coming from different sources, this biped model has a total of 50 DoFs. Motions captured from different human subjects will be dynamically retargeted to this model, directly by our control reconstruction algorithm, and no kinematic retargeting preprocess is done beforehand. Our Asimo-like robot model, shown in Figure 2(b), is $1.2m$

Segment	Mass (kg)	Inertia ($kg \cdot m^2$)
head	5.494	$3.441\times10^{-2}, 2.210\times10^{-2}, 3.441\times10^{-2}$
clavicle & scapula	2.399	$5.062\times10^{-3}, 8.265\times10^{-3}, 8.265\times10^{-3}$
upper arm	1.814	$2.173\times10^{-3}, 9.104\times10^{-3}, 9.104\times10^{-3}$
lower arm	1.526	$1.640\times10^{-3}, 6.753\times10^{-3}, 6.753\times10^{-3}$
hand	0.4588	$3.766\times10^{-4}, 1.597\times10^{-3}, 1.274\times10^{-3}$
trunk	14.31	$1.300\times10^{-1}, 1.601\times10^{-1}, 2.122\times10^{-1}$
pelvis	4.836	$1.620\times10^{-2}, 4.732\times10^{-2}, 4.022\times10^{-2}$
thigh	6.524	$8.714\times10^{-2}, 1.709\times10^{-2}, 8.714\times10^{-2}$
shin	4.612	$5.565\times10^{-2}, 8.931\times10^{-3}, 5.565\times10^{-2}$
foot	1.612	$9.479\times10^{-3}, 9.706\times10^{-3}, 1.714\times10^{-3}$

Table 1: *The dynamic properties of our character model.*

Joint	kp	kd	Strength Scale	Sampling Window	Contact Scale
neck	100	10	1.0, 0.4, 1.0	0.2, 0.2, 0.2	
sternoclavicular	300	30	0.1,1.0,1.0	0.1, 0.1, 0.1	
shoulder	100	5	0.2, 1.0, 1.0	0.2, 0.2, 0.2	3.0
elbow	100	5	0.2, 1.0, 1.0	0.0, 0.0, 0.0	3.0
wrist	20	1	0.1, 1.0, 1.0	0.0, 0.0, 0.0	5.0
waist	1000	100	0.4, 1.0, 1.0	0.2, 0.2, 0.2	
hip	300	30	1.0, 0.2, 1.0	0.4, 0.4, 0.1	
knee	300	30	1.0	0.2	
ankle	100	10	1.0, 1.0, 0.5	0.4, 0.2, 0.1	1.0∼5.0

Table 2: *The PD control parameters and sampling window size (in Radian) for each DoF of each joint of the character model.*

tall and weighs $49.5kg$. Its dynamic parameters can be found in Table 3. We use the triangle meshes rather than geometric primitives for collision detection. This robot model has a total of 35 DoFs.

Note that one joint can have up to three DoFs, and the inertia around different axes are usually not identical. To produce joints that have different strengths in proportion to their associated inertias around different rotation axes, we scale $kp(N/rad)$ and $kd(Ns/rad)$ with the scale factors listed in the fourth column of Table 2. Contact scale is used to increase the stiffness of some weight-bearing joints when in contact with the ground, to propel the body in certain tasks. For example, the ankle joints can be weak for rolling motions, but need to be stronger for running, or else the character may not be able to run forward due to a lack of thrust. Alternatively we can use strong joints all the time, but this can result in stiff-looking motions and is not in accordance with the changing stiffness observable in human muscles and joints.

3.2 Trajectory-based Sampling

3.2.1 Control Representation

We use target poses for PD-servos (Proportional Derivative) to represent motion controls. At every instant of time, the desired angle $\tilde{\theta}$ of a DoF is taken from a desired pose, and the joint torque is calculated as $\tau = k_p(\tilde{\theta}-\theta) - k_d(\dot{\theta})$, to drive the current angle θ towards the desired value. When there is a reference motion, $\tilde{\theta}_i = \tilde{m}_i(t)$ is taken from the trajectory of the corresponding DoF i at the corresponding instant of time t.

Naive tracking of a reference trajectory with PD controllers at the joints is typically unsuccessful for several reasons. First, the captured motions are noisy and sometimes not even physically plausible. Second, the kinematic and dynamic properties of our biped model differ from those of the human subjects from whom the motions were captured. Third, there are various modeling errors associated with a rigid body simulator, including driving an oversimplified rigid biped model with simple PD-servos. We specifically note two problems of PD-servos. One is that the use of constant k_p, k_d parameters is problematic for diversified tasks. For example, the shoulder and elbow joints can be relaxed during a walking motion. In contrast, if the arms are needed during a motion to support the weight of the whole body, they have to be strong and stiff. Therefore we scale the stiffness and damping parameters according to the desired task, as indicated by the last column of Table 2. Another problem of PD-servos is that they are simplified mechanical models of the complex biological neuromuscular actuation systems. They only react to errors and do not produce feedforward torques.

Joint	kp	kd	Strength Scale	Sampling Window	Contact Scale
neck	40	4	1.0	0.0	
shoulder	100	10	0.15, 1.0, 1.0	0.8, 0.8, 0.25	3.0
elbow	100	10	1.0	0.2	3.0
wrist	10	1	0.1, 0.4, 1.0	0.0, 0.0, 0.0	3.0
hip	500	50	1.0, 0.2, 1.0	0.8, 0.1, 0.4	
knee	300	30	1.0	0.6	
ankle	200	20	1.0, 1.0, 0.5	0.6, 0.6, 0.05	1.0∼3.0

Table 3: *The PD control parameters and sampling window size (in Radian) for each DoF of each joint of the robot model.*

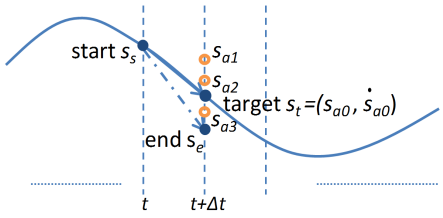
Because of data noise, model discrepancies, and modeling errors, tracking a reference trajectory directly by PD-servos will usually fail to reproduce the desired motion. For example, when tracking a walking motion, the virtual character usually falls within one or two steps. When directly tracking a sideways roll, the character cannot roll more than 40 degrees. It is thus necessary to modify the reference trajectory properly. Hereafter we denote the reference trajectory as $\tilde{m}$, displacements to the reference trajectory as $\hat{m}$, and the simulated motion as m.

3.2.2 Sampling Algorithm and Parallelization

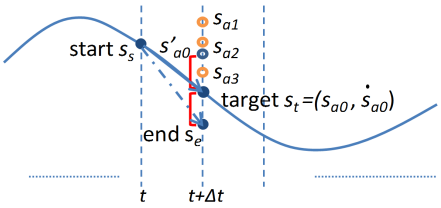
We begin with several definitions. In a multibody system, we define a *pose* as the aggregation of all the internal joint angles and the root orientation at a particular time. A *sample* is defined by a pose displacement, that when added to a pose, forms a new pose. For convenience, we also use *sample* to refer to the displaced new 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.

Given a starting state s_s on the reference trajectory at time t, we now consider possible states of the system at time $t+\Delta t$. The possibilities are illustrated in Figure 3(a). Given a target state s_t on the trajectory at time $t+\Delta t$, we can drive the system towards the pose s_{a0} associated with this target state, using PD-servos. The state resulting from the simulation, s_e, however, is likely to drift away from s_t. Our algorithm then samples in the vicinity of s_{a0} to produce new target poses $s_{a1}, s_{a2}...$ for advancing the simulation, and a series of new end states $s_{e1}, s_{e2}...$ are generated. We can then select the best s_{ek}, the one closest to s_t as measured by a cost function to be described (§3.2.4), and iterate the process using s_{ek} as the start state at time $t+\Delta t$. Progressively advancing the simulation in this fashion will eventually return a simulated motion m that is hopefully close to $\tilde{m}$.

Due to the curse of dimensionality, and the large number of DoFs of our model, we need a sufficient amount of samples at each iteration. We denote the number of samples for one iteration as $nSample$. Because of the existence of noise and model discrepancies, the sample that achieves the closest end state to the reference state at present may not be the best one to select in a longer term. We therefore retain more than one sample at each iteration until the solution process has advanced sufficiently far in order to determine which control samples represent the best control sequence. That is, we save



(a) The sampling process.



(b) The sampling process with feedforward offsets.

Figure 3: *The basic and modified sampling process.*

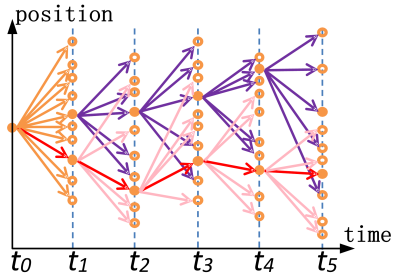


Figure 4: *Schematic illustration of the sampling process with parameters $nIter = 5$, $nSample = 10$, and $nSave = 2$. There are five iterations in total, each using ten samples (orange circles), only two of which are saved (orange dots) and others are discarded. From the two saved samples, ten samples are drawn again. The final chosen path is that which has the lowest cost, shown in red.*

nSave samples at each iteration. This makes the algorithm more far-sighted and improves the robustness with respect to the local minima associated with a constrained unstable system. Each end state associated with the *nSave* samples from the last time step is then used as the start state for the next iteration $nSample/nSave$ times, so that a total of *nSample* samples are tested again. This process is illustrated in Figure 4 with $nSample = 10$, $nSave = 2$, and $nIter = 5$, where *nIter* denotes the total number of iterations.

From the above description of the sampling process, we can see that generating samples and their associated simulations can be done concurrently. Algorithms 1 and 2 describe our parallel implementation of the sampling algorithm. S is an array of length *nSample*, whose elements consist of all the simulation info (s_a, s_s, s_e, c) related to a sample. As before, s_a represents a sample, s_s the start state, s_e its end state, and c is its cost. SM is the two dimensional $nIter \times nSave$ array of the saved samples. From the element of SM at the last time step of a motion, we backtrack *nSave* paths and select the path of minimal cost as the final sample sequence $\hat{m} = (...\hat{s}_{a,t}, \hat{s}_{a,t+\Delta t}, ...)$.

3.2.3 Practical Implementations

The above section introduces the basic sampling algorithm, which does not work well for a high-dimensional non-linear constrained system that is inherently unstable and discontinuous. We develop several critical strategies to improve the robustness and speed of the search algorithm, and the quality of the synthesized motions.

Algorithm 1 : $\hat{m}$ = job_of_master($\tilde{m}, nIter, nSample, nSave$)

Input: reference motion $\tilde{m}$; number of iterations *nIter*; number of samples per iteration *nSample*; number of saved samples per iteration *nSave*

Output: a list of displacement samples $\hat{m}$

```
initialize SM[0]
for t = 1...nIter do
    S = null
    j = 0; k = 0 {initializes counters for sent and received samples}
    while k < nSample do
        while (worker = find_idle_worker()) == null do
            S = S ∪ receive_result_from_worker()
            k++
        end while
        if j <nSample then
            j++
            s_s ← pick_start_state(SM[t − 1]) {picks a start state}
            send_data_to_worker(worker,WORK,s_s,t)
        end if
    end while
    SM[t] =select_samples(S,nSave) {selects only nSave samples}
end for
return m̂ ← search_best_path(SM) {returns the path of minimal cost in SM}
```

Algorithm 2 : calculation_of_worker($\tilde{m}$)

Input: reference motion $\tilde{m}$

```
loop
    (mode, s_s, t)=receive_data_from_master()
    if mode! =WORK then
        break
    end if
    s_a ← generate_sample(m̃, t)
    (m, s_e) ← simulate(s_s, s_a, m̃)
    c ← calculate_cost(m, m̃)
    send_result_to_master(s_a, c, s_e) {sends the sample, the cost, and the end state}
end loop
```

Approximating Feedforward Torques: As mentioned in Section 3.2.1, there are several factors that can cause the simulation to deviate from the target trajectory during direct PD-tracking. The force of gravity is one such factor. If we consider the example of holding an arm in an extended horizontal position, this will require a sustained torque at the shoulder. However, a PD-controller will never reach its desired angle in such a situation because it requires an error in order to generate a torque. It is possible to rely on the sampling to compensate by adjusting the target position upwards, but this is not always practical in our high-dimensional control setting. Instead, we systematically offset s_{a0} by the pose difference between s_e and s_t, and sample around s'_{a0} rather than the original s_{a0} instead. This is illustrated in Figure 3(b), which revises the one-dimensional case shown in Figure 3(a). Stated in another way, we compute an offset to the reference target to approximate feedforward torques, and rely on sampling to generate appropriate torques that compensate for noise in the tracking trajectory and model discrepancies.

In principle, feedforward torques can be computed using inverse dynamics techniques. However, inverse dynamics is extremely challenging for contact-rich tasks due to the unknown contact locations and contact forces. The proposed offset technique greatly improves the success rate of the reconstruction and the quality of the resulted control.

Sampling Time Step: The sampling frequency, determined by Δt, does not have to equal the simulation time step. In fact, it is advantageous to use a lower sampling frequency as long as this does not overly constrain the control that can be applied. A low sam-

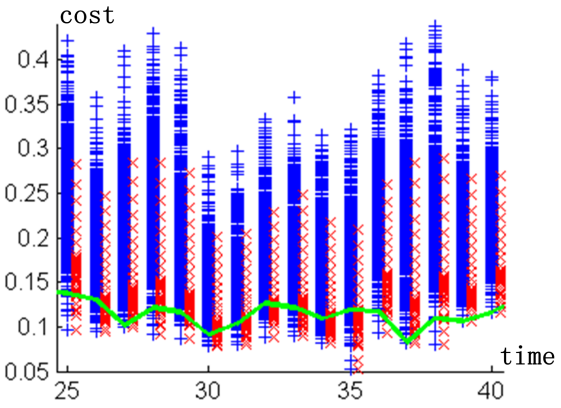


Figure 5: *The cost distribution of samples for part of a walk. Samples tested are shown as a blue +. Saved samples are marked using a red X, offset to the right. The green solid polyline indicates the final minimal-cost path.*

pling frequency speeds up the reconstruction, and results in more compact controls. We notice that low control frequencies, ranging between 5∼60Hz, are also adopted in related animation control methods [Sok et al. 2007; Macchietto et al. 2009]. In this paper, we use $0.1s$ for the sampling time step and $0.0005s$ for the simulation time step. The $10Hz$ sampling frequency is experimentally chosen, but not critical to the success of control reconstructions. In practice, sampling rates lower than $10Hz$ tend to degrade the synthesized motion quality, while higher rates result in reconstructions that are slower to compute. Extremely high or low frequencies do fail, due to nearsightedness or lack of control.

Uniform sampling in time can result in inferior reconstructions for motions that have inherent semantics. For example, in the case of locomotion, extremal limb positions and stance-swing-foot switches are not only visually important, but also mark a change in direction of the ongoing motion or the introduction of a discontinuity. We mitigate the effects of this by segmenting the motion at extremal points and contact switching points, which are detected using a combination of position thresholds and the Kinematic Centroid Segmentation method [Jenkins and Mataric 2003]. Whenever a fixed time step misses a segmentation point, we adjust the several time steps before the critical point to guarantee samples at those moments. The segmentation and time step adjustment are done as a preprocess before the sampling starts.

Sample Selection and Pruning: In advancing the simulation at a given time step, we can ideally generate enough samples to cover the full set of possibilities available to advance forwards from the current state. However, we cannot possibly sample in a uniform, dense fashion because of the curse of dimensionality. We therefore sample randomly from a uniform distribution within a hypercube around the seed. The length of each dimension of the hypercube is determined by several factors. First, noisy data and large model discrepancies demand larger sampling windows. Second, more active joints, such as those actuating contact limbs, usually need larger windows than free-swing joints. Lastly, smooth local dynamics generally only require small magnitude perturbations. We tune the size of sampling windows once, and are able to use the same windows for reconstructing all the motions we have experimented with, as shown in Table 2. However, trial dependent specific tuning, based on the activeness of each DoF for example, can still be useful in improving the quality of synthesized motions.

In order to avoid a search tree that expands exponentially, we develop a strategy to save a representative fraction of the good samples at each iteration. A greedy strategy of saving only the best *nSave* samples can be problematic because locally following the reference trajectory may be misleading in the presence of data noise, model discrepancies, and modeling errors, and result in a local minima. We follow the principle of *diversification* to select samples to save. More specifically, the chosen samples should cover an appropriate range in terms of costs, although samples with lower costs should still be favored. Therefore we first discard samples lying in the top 40% of the cost distribution, and denote the new range of cost as $[cost_{min}, cost_{max}]$. Next, a variable $x = i/nSave, i = 0...nSave-1$ is generated uniformly from $[0,1)$ and is transformed using a convex function $x = f(x)$ into $[cost_{min}, cost_{max})$. We use $f(x) = cost_{min} + (cost_{max} - cost_{min})x^6$. The samples that have the closest cost to x are then saved. In Figure 5, we show the generated samples in blue +, the saved samples in red X, and the final path as a green solid polyline.

3.2.4 Sample Cost Evaluation

The cost of a sample measures its goodness, similar to an objective function in optimization frameworks. We use a weighted sum of four terms shown in Equation 1 as the total cost. Each term will be explained shortly, and example values of the weights are given in Table 4.

$$E = w_p E_p + w_r E_r + w_e E_e + w_b E_b \tag{1}$$

Pose Control: We favor states that are close to the trajectory. There are many distance metrics proposed in the literature to measure the similarity between poses, such as [Kovar et al. 2002]. Here we simply use a weighted squared distance of internal joint angles and angular velocities.

$$E_p = \frac{1}{n}\sum_{i=1}^{n} w_i(d_q(\mathbf{q}_i, \tilde{\mathbf{q}}_i) + 0.1 * d_v(\omega_i, \tilde{\omega}_i)) \tag{2}$$

where $d_v(\omega, \tilde{\omega}) = \|\omega - \tilde{\omega}\|_2$ is the Euclidean distance between two vectors. $d_q(\mathbf{q}, \tilde{\mathbf{q}}) = \|\log(\mathbf{q} \bullet \tilde{\mathbf{q}}^{-1})\|_2$ is the distance between two quaternions, and $\bullet$ represents quaternion multiplication. Quantities with tilde are desired quantities from the reference trajectory. w_i adjusts the relative importance of different joints. We usually just use $w_i = 1$, and adjust the weights to produce more motion variants.

Root Control: The root orientation of underactuated systems cannot be directly controlled by joint torques, but by complex interactions between body parts and the ground. We select samples that closely follow the orientation of the motion capture trajectory.

$$E_r = d_q(\mathbf{q}_{root}, \tilde{\mathbf{q}}_{root}) + 0.1 * d_v(\omega_{root}, \tilde{\omega}_{root}) \tag{3}$$

End-effector Control: Most motions we experiment with involve complicated and delicate interactions between the ground and end-effectors such as hands and feet. Thus the locations of end-effectors are crucial to the success of a task. Here we consider a term that monitors the error between the desired height and the current height of end-effectors, which ensures foot clearance during locomotion for instance.

$$E_e = \frac{1}{k}\sum_{i=1}^{k} d_s(\mathbf{p}_{iy}, \tilde{\mathbf{p}}_{iy}) \tag{4}$$

where k is the total number of end-effectors considered. $\mathbf{p}_i$ is the position of the i^{th} end-effector, and $\mathbf{p}_{iy}$ its y component, i.e., the height of the end-effector. $d_s(p_i, \tilde{p}_i)$ is the absolute difference between two values.

Balance Control: Balance control is normally done 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 a proper landing position. Once the end-effector has contacted the ground, it is harder to change position anymore. Denote the height of the character as h, and the planar vector from end-effector i to CoM as $\mathbf{r}_{ci} = (\mathbf{p}_{CoM} - \mathbf{p}_i)|_{y=0}$, we calculate the balance deviation as follows:

$$E_b = \frac{1}{hk}\sum_{i=1}^{k}(d_v(\mathbf{r}_{ci} - \tilde{\mathbf{r}}_{ci})) + 0.1 * d_v(\mathbf{v}_{CoM}, \tilde{\mathbf{v}}_{CoM}) \qquad (5)$$

3.3 Trajectory-free Sampling for Idle Motions

Idling is common in video games and background movie characters. Although they are usually perceived as easy motions, their quality is surprisingly low compared to more difficult goal-oriented tasks. Several factors contribute to this phenomenon. One is that idling is hard to specify procedurally, because there are no clear goals associated with them, other than a fuzzy feeling that they should look relaxed and non-repetitive. There are no obvious criteria to constrain idling either. For example, self-collision free is usually a requirement for goal-oriented tasks, but idling is exactly the opposite and rich in self-contacts. Capturing idle motions is not popular either. The relatively low importance of idling usually does not justify the high costs associated with motion capture and data processing. Self-contacts and props like chairs also make capture hard. In the CMU motion capture database, we can only find one very short trial of idling in a chair, while in contrast there are dozens of walking motions. Furthermore, the mocap subject sits idle very cautiously in this trial, and does not look relaxed at all. It is indeed problematic for a subject to relax and idle, while wearing a tight suit with 40+ markers on, and being requested to minimize occlusions between markers or rubbing them with each other. Algorithmic studies on idle motions are also very limited. To the best of our knowledge, the only study on idling dates back more than a decade ago, which is a simple application of Perlin noise on canned poses and actions [Perlin 1995].

We advocate using randomized sampling for idle motions. In a broader scope, we could sample configurations that look relaxed, i.e., poses that form multiple self-contacts and contacts with a supporting object, and require low joint torques to maintain. In this work, we restrict ourselves to a more specific scenario where a user specifies a set of key poses, from which our algorithm constructs controllers to drive the character to idle in-between. Unlike control reconstruction from mocap trajectories, here we only have key poses but no reference trajectory. The key challenge for such an algorithm is again the high-dimensionality of the state space. We develop a control planning algorithm based on RRT [LaValle and Kuffner 2000], one of the state-of-art path planning methods, with several crucial modifications:

- In the initialization phase, all the input key poses are simulated with low-gain PD controllers, to generate relaxed poses which form multiple contacts with the supporting furniture. The relaxed poses will replace the original poses as input to RRT.
- During initialization, a start state is directly driven to a target state by PD servos. DoFs that can reach the target are removed from RRT to reduce the dimensionality for sampling.
- In the *EXTEND* operation, control targets are sampled, not from the whole configuration space but from a hyper tube around the spherical linear interpolation of the start and goal states, to limit the sampling space.
- The character is simulated towards a sampled target within a time frame proportional to the distance between states. This step implicitly eliminates undesired collisions but retains valid contacts.

To generate motion variations, we compute multiple trajectories between each pair of key poses. Then as a postprocess, we manually select the best trajectories from all sampled controls, and perform simple algorithmic path simplification and smoothing, similar in spirit to that of [Yamane et al. 2004], but only to the extent where control permits. More specifically, because of the dynamic nature of our planned controllers, there is no guarantee that an edited controller will still work after nodes are removed or smoothed. A controller is tested automatically after every modification, and a failure to reach the target revokes the operation just performed. At runtime, we add a small amount of random noise to the controls. This effectively eliminates zombie-looking fully static poses, and adds more variations to the synthesized motions. Another option here is to use Perlin noise, but we find simple random noise works just fine.

RRT-based path planning has been used for synthesizing manipulation tasks [Yamane et al. 2004]. Their method samples the end-effector configurations and relies on inverse kinematics to solve for joint angles. We directly sample from the pose space. Another difference is that they use RRT in its original kinematic form, while we perform a kind of dynamic RRT where the controls are sampled directly. This partly explains the reasonable quality of our simulated motions even without the velocity profile fitting component of their approach. The lack of stereotypical styles, preferred trajectories, and bell-shaped velocity profiles in idling is likely the other factor that makes velocity profile fitting unnecessary in our case.

4 Results

We use the Open Dynamics Engine (ODE) version 0.11 to simulate our characters. The simulation time step is $0.0005s$ and the coefficient of friction is 0.8. The simulation runs at approximately real-time rates on an Intel Xeon E5520@2.27GHz desktop.

Control Reconstruction: We have reconstructed controls for various tasks, some of which are listed in Table 4. To the best of our knowledge, contact-rich rolling motions have never been considered by previous automatic control reconstruction methods. Table 4 also shows representative reconstruction times of various trials on a small cluster of 80 cores, using 1400 samples for each iteration. The reconstruction is quite robust with respect to noise in input data. Occasionally we have mocap trials that have knees bending backwards severely, or hands flipping around etc. We can still successfully reconstruct control for them, but the simulated motions have these noisy artifacts too, mainly due to the tracking nature of the reconstruction algorithm.

Although the tasks we have tested are different, we are able to reconstruct controls for all of them using one set of sampling and simulation parameters, as shown in Table 2 and Table 4. Manual tuning of these parameters is necessary, but it was not difficult in our experience. We did not find the results to be highly sensitive to specific weighting of the terms in the cost function in Equation 1. We can use the same set of weighting, $w_p = 8, w_r = 5, w_e = 20, w_b = 20$ for example, to reconstruct controls for all the motions. Remaining variations in the given parameters and weights are largely an artifact of experimentation with progressively more diverse motions, with the final set of weights typically being backwards compatible with the original smaller starting set of motions. Although the balance terms are not necessary for tasks where balance does not play an important role, a sideways roll for example. Task-specific tuning of

Trial	Duration	*nIter*	Reconstruction Time	w_p, w_r, w_e, w_b
walk	5.2	62	143	5, 3, 30, 10
run	2.0	27	51	8, 5, 30, 20
sideways roll	3.0	30	78	8, 5, 0, 0
forward roll	3.0	30	78	8, 5, 0, 20
backward roll	2.1	21	57	8, 5, 0, 20
get-up	3.5	40	93	8, 5, 20, 20
kip-up	6.6	66	184	8, 5, 20, 20

Table 4: *Performance statistics: Timing units are in seconds. nIter correlates with the duration of motion. nSample* $= 1400$ *and nSave* $= 200$*. The cluster consists of 10 computational nodes, and each node consists of two Quad-Xeon (E54xx) processors.*

Trial	Duration	*nIter*	Reconstruction Time	w_p, w_r, w_e, w_b
walk	5.2	62	193	5,3,30,10
run	2.0	27	80	8,5,30,20
sideways roll	3.0	30	133	8,5,0,0
forward roll	3.0	30	109	8,5,0,20
backward roll	2.1	21	75	8,5,0,20
get-up	3.5	40	140	8,5,20,20
kip-up	4.3	43	165	8,5,20,20
cartwheel roll	2.1	21	50	2,10,0,0

Table 5: *Performance statistics on the same compute cluster for the Asimo-like robot model. Timing units are in seconds. nSample* $=$ 2000 *and nSave* $= 500$*.*

the weights is possible if the user wishes to emphasize or deemphasize particular terms. For instance, the end-effector term is used to match the height of the feet better for locomotion tasks, but can be disabled for rolling tasks where the user does not care.

nIter correlates with the duration of motion, the sampling time step, and the segmentation method used, as described in Section 3.2.3. When uniform sampling is used, *nIter* equals the duration of motion divided by the sampling time step, which is the case for rolling. For locomotion tasks, semantic segmentation is also used, and that adds more sampling iterations at contact switching and visually important instants of the motion. The contact scale parameters in Table 2 are never used for rolling. Walking uses an ankle contact scale 1.0; running uses an ankle contact scale 5.0; and the get-up motion uses a contact scale 3.0 for the shoulders and the elbows, and 5.0 for the wrists.

Even though the reconstruction algorithm is quite robust, there is no guarantee that every episode of sampling can produce successful controls, due to its randomness. The successful rate is quite high, however, for most trials we have experimented. An input motion with a large feasible region of control, such as the sideways roll, succeeds almost every time. The most challenging trial is one of the get-up motions, where the character fails to stand up two thirds of the time with *nSample* $= 1400$ and *nSave* $= 200$. The original mocap trial actually looks like the subject just made it. However, the failure rate decreases to one fourth if we use *nSample* $= 4000$ and *nSave* $= 400$.

In the accompanying video, we show that multiple runs of the sampling algorithm produce slightly different controls and simulated motions. Note that in [Lau et al. 2009], statistical generative models are learned from multiple trials of the same motion to produce motion variants for walking, swimming etc. We only need one example trial, and can produce physically-plausible motions with rapidly-changing contacts. Furthermore, application of the proposed reconstruction method is not limited to biped models. We use an animation sequence extracted from a wildlife footage [Laurent et al. 2004], and reconstruct a running controller for a Cheetah model. Imaginary characters with non-standard morphology should pose no problem either.

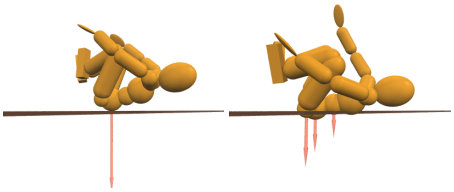


Figure 6: *Rolling on hard floor (left) and soft material (right). Note the difference in contacts and pressure.*

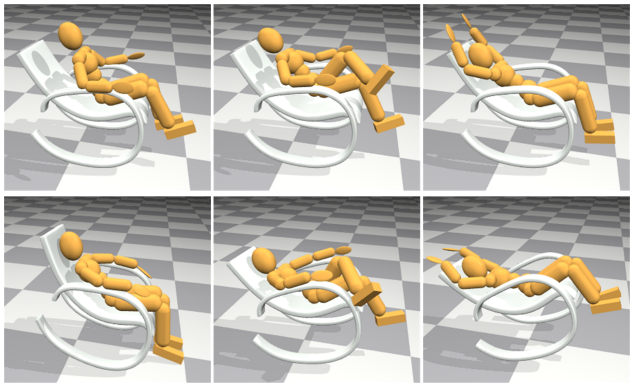


Figure 7: *Keypose-based control construction. Input poses (top) and simulated motions (bottom).*

Motion Transformation: The inherent robustness of the sampling approach enables straightforward physically based motion transformation. Most of the reference mocap moves we use were captured inside controlled lab environments, and we test the algorithm within more challenging settings, or environments with altered physical parameters. For example, we reconstruct a normal walk on a ground with pebbles of random sizes scattered at random locations. The feet adapt to the uneven ground naturally. The same motion is also made to walk on an icy surface with a coefficient of friction 0.1. The feet slide realistically, with the balance automatically maintained by the reconstructed controls. In a balance beam walking test, we manually displace the lower-body trajectory of a mocap walk so that it walks on a straight line. Then we reconstruct the walk on a $10cm$-wide and a $5cm$-wide balance beam, and the sampling succeeds in both cases with correct balancing behaviors added. These transformed motions are likely difficult to be captured or manually designed.

Without any difficulty, we can reconstruct the forward rolling with a ramp of 10 degrees, or a $10cm$ height raise, or a $50cm$ height drop put in front of the character. More adverse conditions, such as a $100cm$ height drop shown in Figure 1(a), do not necessarily defeat the reconstruction. But the character plunges too fast and lands on its back in the synthesized motion, whereas in real life a martial artist would control the timing of the fall to land on his hands first. This problem is caused by the tracking nature of our reconstruction. On the other hand, if simple editing of the reference trajectory is affordable, we can easily slow down the diving part of the input data so that the reconstructed control can land the virtual character on its hands.

Sometimes it is desired to be able to inspect the distribution of contacts and pressure during the course of a motion, for martial arts education or biomechanical applications such as injury analysis and product design [Payton and Bartlett 2007]. We can reproduce the movement of contacts and the change of pressure by simulating from reconstructed controls, as shown in Figure 6. We further re-

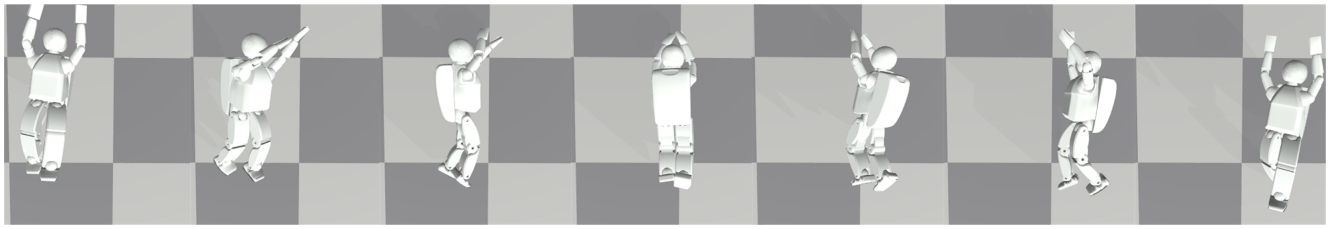


Figure 8: *A sideways roll captured from human retargeted to an Asimo-like robot.*

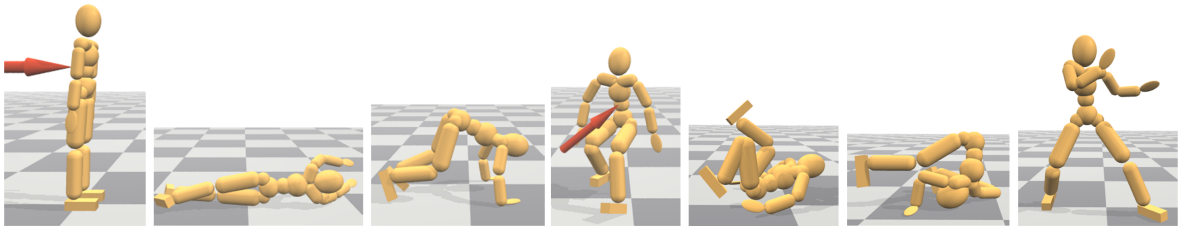


Figure 9: *An example interaction with the virtual character. The character gets pushed and falls, rolls sideways, gets up, gets pushed again, rolls backwards, and gets up again.*

construct controls for the roll on a springy crash pad, simulated by setting the error reduction parameter to 0.8 and the constraint force mixing parameter to 0.007 in ODE. The two snapshots in Figure 6 are taken at the same instant of the roll. As we can see, the rolling on the soft material lags behind a little, with more contact points each of less pressure. We speculate that the sampling algorithm will also be effective in a real deformable body simulator.

Motion Retargeting: Motion retargeting usually refers to the process of editing an existing motion for a new kinematic model so that kinematic constraints and effects are maintained or meaningfully adapted. Our control reconstruction process can achieve *dynamic* motion retargeting, where dynamic effects as well as kinematic constraints are adapted. We retarget various motions captured from human subjects to the Asimo-like robot model shown in Figure 2(b). These retargeting tasks are extremely challenging because of the huge differences between the human subjects and the Asimo model, in terms of their kinematic parameters, dynamic parameters, and collision detection geometries. Nonetheless, the sampling-based algorithm succeeds in reconstructing controls for most of the motions, such as the cartwheel shown in Figure 1(b) and the barrel roll shown in Figure 8. The performance statistics is given in Table 5.

The root positions of the original mocap data are simply linearly scaled down for Asimo using the ratio of body heights, so there are many severe ground penetrations and floating moves in the reference trajectories. However, we are able to eliminate these in the synthesized motions, without any kinematic retargeting preprocess, although the synthesis quality might be degraded because of this. Compared to reconstructions on the character model, the controls reconstructed on the Asimo model require larger sampling windows, produce motions of lower quality, and are 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 quite 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 able to successfully reconstruct controls for the backward roll on Asimo. We suspected that it is because Asimo only has a one-DoF neck joint. We temporarily assigned three-DoFs to Asimo's neck, and then are able to produce a successful backward roll from the sampled controls.

Idling: We test the trajectory-free control construction with an idling-in-chair scenario. A few input example poses, such as the top row of Figure 7, can generate relaxing motions with subtle movements. In the accompanying video, we compare the sit idle motion synthesized from our algorithm, an artist designed animation clip, and a motion capture trial. We deem the quality of the synthesized motion comparable, if not better than, the quality of idling animations from the other two sources. In many cases people have thought our synthesized idling was motion captured. We encourage the readers to watch the accompanying video and evaluate the quality yourself. In addition, with the constructed controls, we can easily simulate dynamic effects such as rocking back and forth in a rocker, shown in the bottom row of Figure 7.

Motion Composition: To support interactive applications such as video games, we need to compose controls individually constructed [Faloutsos et al. 2001]. However, our reconstructed controls are open-loop in nature. That is, for each reference motion $\tilde{m}$, the reconstructed control produces a simulated trajectory m, called the canonical trajectory from now on. When starting from a state not on the canonical trajectory, either as a result of external perturbations or changed environments, the virtual character is unlikely to follow m to accomplish the planned task. Previous methods for constructing feedback laws are either task-dependent, or not robust enough for our target application [Yin et al. 2007; Muico et al. 2009]. Another possibility is to construct dense control policies as in [Sharon and van de Panne 2005; Sok et al. 2007], yet the memory requirement of these systems, when applied to many tasks as in our case, is too high for console games.

We thus use a hybrid approach to transition between controls. Within every m, we select cut points where other controllers can transition into. When there is no disturbance, motions are simulated or transitioned dynamically as planned. Upon external perturbations, the current controller still acts as planned for another 200ms, to approximate the reaction time in biological systems. Then the perturbed state, most likely far away from m already, is compared with all the cut points of all the controllers. If a close cut point exists, we simply switch to simulate from the matching cut point, and kinematically blend the end state resulted from the perturbation into the new canonical trajectory. If no close cut point can be found, a semi-ragdoll controller takes over, and produces simple reflex-like behaviors, such as arm extensions, for fall protection. Transitions can still be made anytime during the ragdoll controller, if a suitable

cut point can be found. An example would be a transition into a backward roll during the course of a backward fall. In most cases, however, the character just collapses to the ground, and looks for a proper get-up controller to transition into. PD control is then used to drive the character close to the canonical trajectory of the chosen get-up controller, which will then take over.

Figure 9 is one example interaction with our virtual character. Note that when the character is acting as planned, we can in theory just kinematically play back the captured motions, similar to the approach of [Zordan et al. 2005]. An extra kinematic retargeting process is needed, however, to transform all the data captured from people of different sizes and proportions to the same biped model. In addition, we choose to always simulate, to avoid the constant switching back and forth between a kinematic controller and a dynamic one.

5 Discussion

We introduce the use of randomized sampling to tackle control problems for contact-abundant motions such as rolling and idling. We also present practical design choices and efficient parallel algorithms to realize an interactive, workable algorithm. As with other high dimensional problems, the specific design choices related to representation and implementation play a critical role in developing a working system. The robustness and flexibility of the scheme are demonstrated through physics-based motion transformation and retargeting, and on different kinematic and dynamic models.

Scalability: The proposed algorithm scales almost linearly with respect to the number of available cores. With our 80-core computer cluster, the current implementation requires approximately $25s$ of wall-clock time to reconstruct $1s$ of motion. With an additional order of magnitude of increase in the number of cores, an artist can potentially work with the system at interactive speed. This may be realized with the help of architectures such as the Amazon Elastic Computing Cloud, which allow for the dynamic allocation of large scale computing resources.

Robustness of Reconstruction: Our method is not an optimization, so there is no convergence problem. The algorithm returns within a fixed amount of time, given a set of sampling parameters and a particular motion. The reconstructed control is not guaranteed to be successful, however, after a single run of the sampling algorithm. A failure means falling while walking, unable to turn over while rolling etc. Our experiments show high success rates though. Roughly speaking, $> 80\%$ of the reconstructions finish with a success rate of $> 80\%$. Furthermore, because each reconstruction run requires only minutes, the algorithm can be run multiple times on the same problem to allow a user to explore different possible reconstructions. Another option is to increase the number of drawn samples and saved samples at each iteration.

We have succeeded for all the motions we tried with the human model. In retrospect, the rich contacts probably help us in some ways in counteracting drifts and errors. It would be interesting to try our method on aerial motions where we cannot do anything about drifts for long durations. In the future, we also wish to add a backtracking ability to the sampling algorithm. This would help preclude making decisions that appear to be good in the short term but turn out to be bad ones later on. In case offsprings of bad samples have driven away all descendants of good samples during sample pruning, it would be desirable to rewind to previous iterations and resample to make up the mistake. This strategy is likely to further improve the success rate of reconstruction.

Robustness of Control: Our 'control bases' are fixed in time, meaning that control actions are time-dependent but not state-dependent. Put it another way, the controls we construct are open-loop. Although this provides a solution to inverse dynamics and motion transformation for contact-abundant motions, the controls are not robust with respect to external perturbations or environmental changes. We can try to search for dense control policies, i.e., mappings between states to actions, as in [Sharon and van de Panne 2005; Sok et al. 2007]. But this is likely difficult because of the existence of a myriad of local minima caused by many fast-changing contacts. A more promising avenue is to build task-dependent, active feedback mechanisms to form close-loop controllers as in [Yin et al. 2007]. This may or may not be possible for certain types of tasks.

Generalization: One of the major motivations for physics-based motion synthesis techniques is to generalize motion capture data [Zordan and Hodgins 2002; Yin et al. 2003]. We have demonstrated several forms of motion generalization within the same framework, and it is interesting to think about how to push them even further. (a) 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. (b) motion variation: We focus on small variations that can be treated as noise, similar to what is modeled in [Lau et al. 2009]. That is, the same person in the same physical and mental conditions still cannot reproduce his last motion exactly. For motion variations caused by other reasons, such as mood changes or physical injuries, new mechanisms have to be devised. (c) 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, we recommend using a controller adaptation scheme, such as [Yin et al. 2008], on the reconstructed controls rather than hold on to the reference trajectory all the time. Manual editing of the input trajectories can also help shape the synthesized motions. (d) motion retargeting: The Asimo model differs significantly from the human model, and if the model were to differ more, at some point the retargeting would simply fail. Continuation methods may be helpful for more aggressive retargeting.

Smoothness of Motion: In our method, samples are independently drawn and there is currently no mechanism to control the smoothness of the synthesized motion. This can result in the introduction of noise into the synthesized motion. It is not problematic for motions such as rolling because of the changing pattern of collisions, but can be noticeable for other classes of motions, such as the free-swinging arms in a balance beam walk. In such cases we propose using a butterworth filter to post-process the generated motion. Another alternative would be to add a smoothness term when evaluating the sample cost.

Acknowledgements: We would like to thank the anonymous reviewers for their constructive comments. We thank all our motion capture subjects, students who helped post-process the data, and organizations and authors who generously shared their motion capture data. We thank Xiao Liang and Teng Gao for helping us with coding on the HPC cluster; Kevin Loken for helpful advice on a number of occasions.

References

BARBIČ, J., AND POPOVIĆ, J. 2008. Real-time control of physically based simulations using gentle forces. *ACM Trans. Graph. 27*, 5.

BRUBAKER, M. A., SIGAL, L., AND FLEET, D. J. 2009. Estimating contact dynamics. In *IEEE International Conference on Computer Vision (ICCV)*.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 219–228.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22*, 2, 182–203.

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

HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 153–162.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, 879–887.

JENKINS, O., AND MATARIC, M. 2003. Automated derivation of behavior vocabularies for autonomous humanoid motion. In *2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

JORDAN, M. I., AND WOLPERT, D. M. 1999. Computational motor control. In *The Cognitive Neurosciences*, M. Gazzaniga, Ed. MIT Press.

KAVRAKI, L. E., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation 12*, 4, 566–580.

KAWATO, M. 1999. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology 9*, 718–727.

KIM, M., HYUN, K., KIM, J., AND LEE, J. 2009. Synchronized multi-character motion editing. *ACM Trans. Graph. 28*, 3.

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

LAU, M., BAR-JOSEPH, Z., AND KUFFNER, J. 2009. Modeling spatial and temporal variation in motion data. *ACM Trans. Graph. 28*, 5.

LAURENT, F., LIONEL, R., CHRISTINE, D., AND MARIE-PAULE, C. 2004. Animal gaits from video. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 277–286.

LAVALLE, S. M., AND KUFFNER, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Proceedings Workshop on the Algorithmic Foundations of Robotics*.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2006. Composition of complex optimal multi-character motions. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 215–222.

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

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

NGO, J. T., AND MARKS, J. 1993. Spacetime constraints revisited. In *Proceedings of SIGGRAPH 1993*, 343–350.

PAYTON, C., AND BARTLETT, R. 2007. *Biomechanical Evaluation of Movement in Sport and Exercise*. Routledge.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1, 5–15.

SHARON, D., AND VAN DE PANNE, M. 2005. Synthesis of controllers for stylized planar bipedal walking. In *ICRA05*, 2387–2392.

SIMS, K. 1994. Evolving virtual creatures. In *Proceedings of SIGGRAPH 1994*, 15–22.

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

TSIANOS, K. I., SUCAN, I. A., AND KAVRAKI, L. E. 2007. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review 1* (August), 2–11.

TWIGG, C. D., AND JAMES, D. L. 2007. Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph. 26*, 3.

VAN DE PANNE, M., AND FIUME, E. 1993. Sensor-actuator networks. In *Proc. ACM SIGGRAPH*, ACM, 335–342.

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

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

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Proceedings of SIGGRAPH 1988*, 159–168.

YAMANE, K., KUFFNER, J. J., AND HODGINS, J. K. 2004. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph. 23*, 3, 532–539.

YIN, K., CLINE, M. B., AND PAI, D. K. 2003. Motion perturbation based on simple neuromotor control models. In *PG'03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 445–449.

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

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

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *ACM SIGGRAPH Symposium on Computer Animation*, 89–96.

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