Online Motion Synthesis Using Sequential Monte Carlo

Perttu Hämäläinen[1*] Sebastian Eriksson[1] Esa Tanskanen[1] Ville Kyrki[1] Jaakko Lehtinen[1,2]

[1] Aalto University [2] NVIDIA

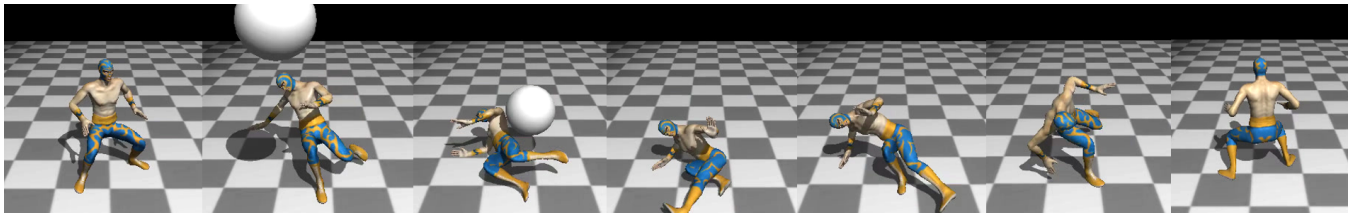

Figure 1: *An example of the synthesized animation (downsampled from the original 30 fps). Frame 1: balancing in the user-specified ready stance. Frames 2,3: The character anticipates that the ball would hit it and dodges down. Frame 4: anticipation pose to get enough leg swing momentum. Frames 5,6,7: swinging the leg around and following with the rest of the body to end up again in the ready stance. The ready stance facing direction was not given as a goal.*

Abstract

We present a Model-Predictive Control (MPC) system for online synthesis of interactive and physically valid character motion. Our system enables a complex (36-DOF) 3D human character model to balance in a given pose, dodge projectiles, and improvise a get up strategy if forced to lose balance, all in a dynamic and unpredictable environment. Such contact-rich, predictive and reactive motions have previously only been generated offline or using a handcrafted state machine or a dataset of reference motions, which our system does not require.

For each animation frame, our system generates trajectories of character control parameters for the near future — a few seconds — using Sequential Monte Carlo sampling. Our main technical contribution is a multimodal, tree-based sampler that simultaneously explores multiple different near-term control strategies represented as parameter splines. The strategies represented by each sample are evaluated in parallel using a causal physics engine. The best strategy, as determined by an objective function measuring goal achievement, fluidity of motion, etc., is used as the control signal for the current frame, but maintaining multiple hypotheses is crucial for adapting to dynamically changing environments.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation, motion synthesis, motion planning, sequential Monte Carlo, particle filter, optimization

Links: DL PDF WEB VIDEO

*e-mail:first.last@aalto.fi

1 Introduction

Production of 3D character animation is a slow, laborious process. Further, if one aims for expressive interaction and realism, the amount of animation required in interactive software like games is practically infinite. A long line of research addresses these problems by seeking to transform the animator or game designer into a *choreographer* who commands virtual agents that algorithmically synthesize the desired motions based on high-level goals. Successful synthesis results in physical validity (realistic body part masses and muscle forces, respecting non-penetrating contacts and friction), and leads naturally to movement qualities like "squash-and-stretch" and anticipation [Witkin and Kass 1988; Lasseter 1987].Following the seminal work of, e.g., Witkin and Kass [1988] and Sims [1994], basic behaviors such as balancing and locomotion can now be generated in real-time, and offline systems exist for synthesizing more complex motions [Geijtenbeek et al. 2011; Al Borno et al. 2013; Erez et al. 2013]. However, online, interactive synthesis of difficult, contact-rich movements, such as acrobatics, remains a challenge, particularly in unpredictable dynamic environments where prior animation or motion capture data is unavailable.

This paper tackles the problem using a novel approach based on Sequential Monte Carlo (SMC) methods for multimodal tracking, here applied to trajectory optimization and Model-Predictive Control (MPC). We present a trajectory optimization system with two key design goals: 1) the resulting movement should be creative and interesting with minimal input data, i.e., goals and constraints instead of pre-made animation or motion capture data, and 2) the system should operate at an interactive frame rate at design time, enabling rapid iteration of the goals and constraints. The output of our system is a time-varying control strategy that drives the character towards the specified goals, while accounting for changes in the environment. Furthermore, the output can be mapped to a more lightweight runtime controller using standard machine learning techniques.

We score the potential control strategies by an objective function (a fitness function) that measures goal attainment and the physical properties of the motion. The function is highly non-convex and multimodal, reflecting the fact that many strategies may lead to the desired goal. Naturally, some are better than others — smoother, use less energy, "more natural"; however, finding the

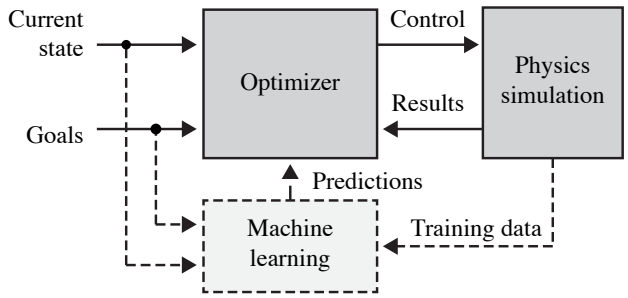

Figure 2: *High-level system architecture.*

global maximum using standard nonlinear optimization is not a robust approach, since changes in the environment may unpredictably change the objective function. To attain robustness in the face of this uncertainty, we maintain a discrete family of potential control strategies. Formally, treating each control strategy as a point in a high-dimensional space (to be made explicit below), we evolve a population of samples using Sequential Monte Carlo sampling so that the ensemble remains well-distributed even when the fitness landscape changes. This allows the optimizer to switch strategies if changes in the environment so dictate. Our formulation also allows straightforward parallelization: the objective function values for the samples can be computed in an arbitrary order. We further exploit temporal coherence by forming a sample generation prior for the current frame based on previous frames.

Our contributions are

- the introduction of SMC to online synthesis of physically valid character motion;
- a novel sequential sampling method that allows easy integration of machine learning. The sampler utilizes kD-trees for adaptive sampling;
- online, near-real-time synthesis of complex get up strategies, e.g., planting a hand on the ground, leaning on the hand to allow moving a foot closer, and finally shifting weight on the foot to rise up.

An example of the motion generated by our system is shown in Figure 1. Our character is able to balance in a given pose, dodge projectiles, and improvise a variety of complex get up strategies if forced to lose balance, all without precomputation or training data.

Figure 2 gives an overview of the main components of our system, including the multimodal sampler/optimizer that generates motion plans, a parallelized physics engine that is used to simulate the movement resulting from each motion plan, and an optional machine learning system that generates one or more predictions used for seeding the adaptive sampling in each frame.

2 Related work

Physically Valid Procedural Character Animation The vast research on procedural character animation is challenging to review thoroughly within the scope of this paper, as central work such as spacetime constraints by Witkin and Kass [1988] has hundreds of citing papers. For a more complete review, we refer to Geijtenbeek et al. [2011]. We focus on optimization based animation of active characters (e.g., locomotion, jumping, dodging) instead of passive ragdolls that are trivial to implement using off-the-shelf physics engines like Open Dynamics Engine or Bullet. We do not discuss procedural animation techniques such as parametric motion graphs [Heck and Gleicher 2007] that enable goal-driven behavior based on a library of animation data, but do not enforce physical constraints such as non-penetrating contacts. Such techniques are covered, e.g., in the review by Pejsa and Pandzic [2010].

Offline Optimization The problem of synthesizing diverse and physically valid motion based on spacetime constraints (e.g., jump and land in a specific pose at a specified time while minimizing energy expenditure) has largely been solved in the offline case. Much of the work has focused on extensions of the quadratic programming (QP) formulation of Witkin and Kass [Witkin and Kass 1988; Cohen 1992; Fang and Pollard 2003; Safonova et al. 2004], where the optimized variables include the root position and rotation, and joint rotations for each animation frame. QP is well suited for spacetime optimization, as target poses can be defined as equality constraints, contacts as inequality constraints, and energy minimization and smoothness can be included in the quadratic cost. However, the synthesized motions have been limited by the need for prior knowledge of contact information, such as in what frames the character should touch the ground and with which body parts. This limitation was overcome by Mordatch et al. [2012], who introduced auxiliary optimized variables that specify the contact information, and who used L-BFGS for optimization.

Motion Synthesis as a Control Problem Spacetime optimization can also be approached as a control problem. In this case, the optimized variables describe the evolution of control parameters such as joint torques over time, and the resulting motion is computed by forward dynamics simulation [Ngo and Marks 1993; Wampler and Popović 2009; Al Borno et al. 2013]. This way, the problem falls in the domain of MPC. Control optimization has the benefit that physics constraints such as continuity and contacts are handled frame-by-frame by the physics engine and do not have to be included in the optimization. The approach also handles additional dynamic objects, whereas a direct Witkin and Kass -style spacetime formulation needs additional variables for each moving object. On the other hand, symbolic expressions of Jacobians and Hessians are not available, which motivates the use of stochastic, derivative-free optimization methods. The dynamics simulations for evaluating each sampled control vector are costly but straightforward to parallelize. In light of the recent work by Mordatch et al. [2012] and Al Borno et al. [2013], both deterministic spacetime optimization and stochastic derivative-free control optimization appear equally suitable for offline synthesis of contact-rich, complex and acrobatic motions.

Online Optimization and Control using Prior Data Considering online synthesis of physically valid motion at interactive frame rates, there are various approaches. Offline optimization has been used to learn the parameters of neural networks and other controller types that can be used in real-time physics simulation [Sims 1994; Reil and Husbands 2002; Geijtenbeek et al. 2013]. This has the drawback of limited generalization to novel situations. Reference motions can be tracked under disturbances, e.g., using an MPC approach with a short-horizon QP controller that is run intermittently and augmented with a proportional-derivate (PD) controller at each simulation step [Da Silva et al. 2008], or preprocessing motion data with spacetime optimization and then using a nonlinear quadratic regulator [Muico et al. 2009], or sampling-based optimization of a time-varying pose displacement relative to the reference trajectory [Liu et al. 2010]. Existing controllers can also be combined to form novel controllers for new goals [da Silva et al. 2009].

Online Optimization Without Prior Data Operating without reference motions or controllers complicates online synthesis. Our

work falls into this category, and we draw inspiration from two main prior systems.

Jain et al. [2009] have implemented balancing, stepping and other behaviors using a frame-by-frame QP optimizer augmented with a state machine that breaks movement down into subgoals that can be implemented with a planning horizon of a single frame. The subgoals are quite detailed, such as moving a lifted foot so that the ground projection of the character's center of mass lies midway between the feet. This raises concerns about robustness, e.g., to obstacles for foot placement, and decreases the probability of creative, emergent movements. In general, there is a trade-off between minimizing computational cost (a short planning horizon), and minimizing the amount of prior information and assumptions needed in the form of motion data or a state machine definition. Our system does not need motion data, and our optimizer automatically generates balancing and footsteps without predefined states thanks to a planning horizon of up to 4 seconds, which is enough for completing a rebalancing step, or even rolling on the ground and bouncing back up. Our approach is also inherently predictive — characters can anticipate events without hand-coded prediction algorithms.

Our work is perhaps closest to Tassa et al. [2012] who also studied the actions of balancing and getting up, and used a multithreaded physics engine to forward-simulate candidate trajectories. We extend their approach in three key areas: we use a longer planning horizon (up to 4 seconds vs. their 0.5s), simultaneously track multiple modes of the fitness function (their iLQG method is unimodal), and use a more complex character model, including 3-DOF joints. As a result, our system adds the ability to plan movements with several phases — e.g. getting up by planting a hand, pushing with the hand to allow moving a foot closer, and then shifting weight on the foot, as shown in Figure 14. The character of Tassa et al. is able to get up from a lying position in a single bounce, implying rather loose limits on the control torques, which simplifies the planning problem. In later work, the problem of springing straight up was solved by designing a state machine that explicitly breaks down the task into a sequence of subtasks [Erez et al. 2013].

Sequential Monte Carlo Sampling SMC has been used widely in various tracking problems [Arulampalam et al. 2002; Doucet and Johansen 2009]. Body tracking using computer vision is especially close to our work, as many tracking systems feature both particle filters (a form of SMC) and articulated human body models [Deutscher et al. 2000; Schmidt et al. 2006]. SMC has also been recently introduced to control optimization [Stahl and Hauth 2011; Kantas et al. 2009; de Villiers et al. 2011], but to the best of our knowledge, it has not been applied to motion synthesis with complex articulated characters. Although our sampler bears similarities, e.g., to the particle filter variants discussed by Arulampalam et al. [2002], it is more precisely a sequential version of the mutated kD-tree importance sampling of Hämäläinen et al. [2006], which in turn is based on the hierarchical subdivision sampling of Kajiya [1986]. Compared to particle filters, we apply similar prediction, weight updating, and resampling operations to the samples, but the formulae differ as the tree structure is used to compute sample weights and adaptive exploration variances.

In addition to Hämäläinen and Kajiya, many others have combined kD-trees with sampling. For example, Thrun et al. [2000] describe Bayesian mobile robot localization using a kD-tree for sample weight computing, and for conditional sampling from a precomputed model of the joint distribution of poses and observations. Rudoy and Wolfe [2006], building on Ihler et al. [2003] describe efficient tree-based sampling from products of Gaussian mixtures.

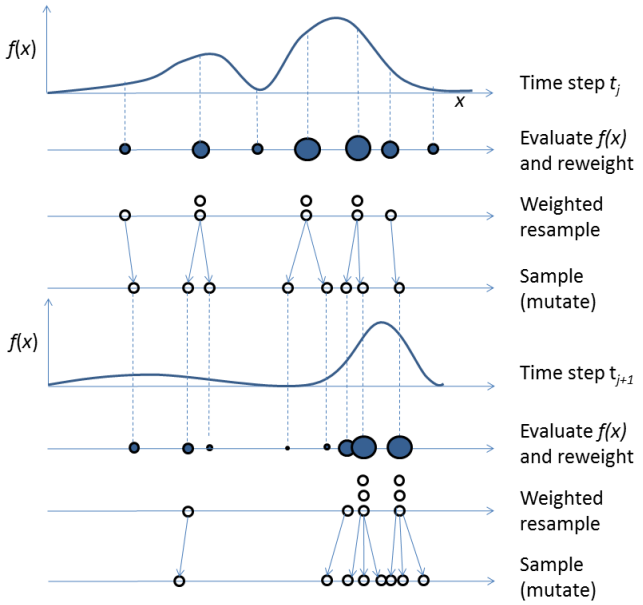

Figure 3: *An illustration of the basic principles of SMC. The tracked probability density f(x) is approximated by a set of samples, which are iteratively weighted, resampled and sampled. In the sampling step, new samples are drawn from proposal densities based on the previous samples. This is analogous to how many stochastic optimization methods mutate samples to explore the parameter space. During the resampling step, the samples at the peaks of f(x) produce more "offspring", while others may die out. Note that the weights depicted here are only exemplary; the exact weighting formulae vary between methods.*

3 Adaptive Sequential Importance Sampling

3.1 Overview

We seek to control a physical character towards attaining goals. We formulate this as finding the global maximum of a real-valued non-negative objective function (fitness function) $f(\mathbf{x}; t)$, where the vector $\mathbf{x} \in \mathbb{R}^k$ defines a control strategy represented as time-varying target joint angles and other parameters, explained in detail in Section 4.2 [1]. Time t is a parameter rather than a domain variable, and it accounts for a dynamic environment. As the search space consists of an approximation to all the possible ways to drive the actuators over a couple of seconds, it is easy to appreciate that the objective function is multimodal, and that the modes shift, appear and vanish with time.

Multimodality of the objective function motivates a Sequential Monte Carlo (SMC) approach. The core idea of SMC methods is that a sequence of target probability densities is approximated using an evolving set of weighted samples, as illustrated in Figure 3. The sample set can then be used for estimating the modes of the density function. A thorough mathematical treatment can be found, e.g., in [Arulampalam et al. 2002; Doucet and Johansen 2009].

Although the terms particle filters and SMC are sometimes used interchangeably, we use the latter in the sense of Doucet and Johansen [2009] to denote a general class of algorithms for sampling

[1] All our parameters have minimum and maximum values, and the problem is thus inequality-constrained by axis-aligned planes, i.e., the space of possible solutions is a hypercube.

from a sequence of target probability densities. Contrary to particle filter approaches, we simply treat the objective function $f(\mathbf{x};t)$ as a sequence of unnormalized target probability densities, instead of modeling the posterior density $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ or the marginal likelihood $p(\mathbf{y}_{1:t})$, where $\mathbf{y}$ denotes observations related to $\mathbf{x}$.

Doucet and Johansen [2009] show that particle filters can be interpreted as special cases of a generic SMC algorithm, which however requires all samples to be drawn from known proposal densities. Our sampler avoids this requirement, allowing the insertion of additional arbitrarily obtained candidate samples, e.g., initial guesses from the machine learning component in Figure 2. Whereas typical SMC weighting comprises a division by the proposal density, we use a kD-tree-based estimate of the realized sample density. This provides a unified way of weighting random samples with known proposal densities and initial guesses with no proposal density. Additionally, the kD-tree provides means for adapting the search variance so that samples at low fitness regions are perturbed more.

Algorithm Overview We maintain a population $\{\mathbf{x}_i, f(\mathbf{x}_i;t)\}$ of N samples and their associated fitnesses that evolve over time. For each frame, the algorithm performs the following steps:

1. Prune the sample set. Keep only the best M.
2. Draw a set of K new samples by optional heuristics and machine learning predictions that may depend on the current state.
3. Construct a *sampling prior* $q(\mathbf{x})$ based on the $M+K$ samples by inserting the samples in a kD-tree and constructing an adaptive PDF.
4. Until the budget of N samples is reached, draw new samples from $q(\mathbf{x})$. Use each new sample to adaptively update the prior $q(\mathbf{x})$.[2]
5. Pick the best sample and use it for driving the simulation forward for the current time step.

The construction of the sampling prior $q(\mathbf{x})$ and its adaptive refinement are described in Section 3.3. Details on the outer loop are presented in Section 3.4. The heuristics and machine learning component are detailed in Section 4.5.

3.2 A 2D Example

To provide visualizations and intuition on our sampler applied to motion optimization, we present a 2D problem with nonlinearities and multimodality arising from physical contacts. Figure 4 gives an example of the objective function of 2D ball throw optimization. The optimized parameters are $\mathbf{x} = [s, \alpha]^T$, where s is the throwing speed, α is the throwing angle, and the aim is to get the ball as close to a specified target $\mathbf{g}$ as possible. Figure 4A illustrates the physical setup and shows two trajectories resulting from different $\mathbf{x}$. The target $\mathbf{g}$ is shown as a black circle.

We illustrate two different objective functions: the first, illustrated in Figure 4B, computes the closest point $\mathbf{c}(\mathbf{x})$ of the trajectory and the target without regard for timing, i.e., $f(\mathbf{x}) = \exp\{-\|\mathbf{c}(\mathbf{x}) - \mathbf{g}\|^2\}$. This produces a landscape with ridges, each ridge corresponding to, e.g., different number of bounces. The two example trajectories are marked by the red and green circles.

The second goal, illustrated in Figure 4C, aims to hit the target at a specified point in time. The corresponding objective function simply evaluates the distance to the target at this time in the trajectory. Now, some of the ridges become peaks, but the landscape is still multimodal, as the ball can still reach the target using a variety of different bounce sequences. Figure 4D illustrates the adaptive kD-tree that defines the sampling prior $q(\mathbf{x})$, whose construction is detailed in the next section.

[2]This iteration of sampling from a model and updating the model based on the samples is also central to some other optimization methods, such as the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which has been gaining popularity in the motion synthesis literature. However, whereas CMA-ES updates a unimodal model (a single Gaussian), our model is multimodal.

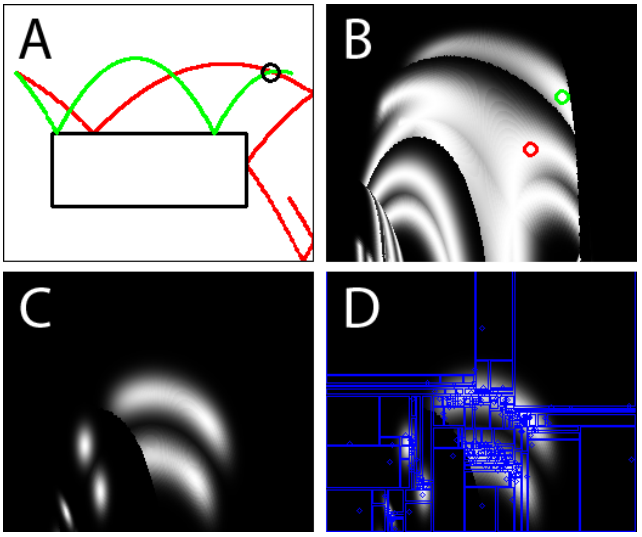

Figure 4: *A) A 2D ball throw test scene. The black circle denotes the throw target. Two example ball trajectories are shown in red and green. B) The objective function mapped to image intensity, with respect to throw angle (horizontal axis) and throw speed (vertical axis). The green and red circles show the location of the example trajectories in the parameter space. C) The objective function landscape when the time to hit target is constrained. D) An example of 100 samples and hypercubes (rectangles in this 2D case) generated by one time step of Algorithm 2.*

Algorithm 1 Adaptive Importance Sampling using a kD-tree.

Draw $\mathbf{x}_0$ uniformly in parameter space, evaluate $f(\mathbf{x}_0)$
$root \leftarrow \{\mathbf{x}_0, f(\mathbf{x}_0)\}$
repeat
  Randomly select leaf node i with probability $\propto w_i$
  Draw a sample $\mathbf{x}_{new} \sim \mathcal{N}(\mathbf{x}_i, \mathbf{C}_i)$
  Evaluate $f(\mathbf{x}_{new})$
  $\{n_1, n_2\} \leftarrow$ INSERTTREE($\mathbf{x}_{new}$) ▷ n_1, n_2 are new leaves
  $w_{n_1} \leftarrow V_{n_1} f(\mathbf{x}_{new})$ ▷ n_1 is where $\mathbf{x}_{new}$ ends up
  $w_{n_2} \leftarrow V_{n_2} f(\mathbf{x}_{n_2})$ ▷ n_2 contains previous sample
until #samples $= N$

3.3 Adaptive Importance Sampling Using a kD-tree

We first describe an adaptive importance sampler for a *time-invariant*, multimodal, unnormalized objective function $f(\mathbf{x})$. The method was proposed in [Hämäläinen et al. 2006], but we repeat it for completeness, and then extend it for the sequential, time-varying case. The process is outlined in Algorithm 1 and illustrated in Figure 5 and Figure 6. The process draws samples approximately following $f(\mathbf{x})$, allocating more samples at regions where $f(\mathbf{x})$ is high, with the approximation of $f(\mathbf{x})$ gradually improving

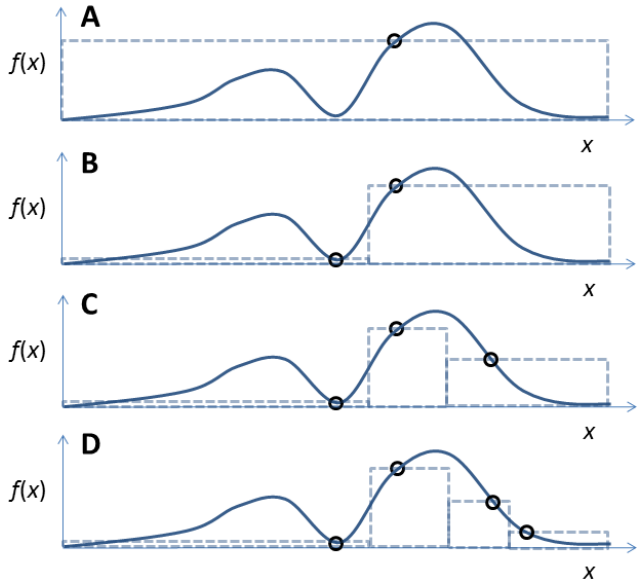

Figure 5: *1D illustration of sampling from f(x) according to Algorithm 1, but generating the samples inside the selected hypercubes instead of drawing them from the normal distribution. The circles denote samples and the rectangles illustrate the adaptive space subdivision into hypercubes. In 1D, rectangle widths are the hypercube volumes, and rectangle areas correspond to sample weights in Algorithm 1. In step B, the sample falls in a valley of $f(x)$, which then biases the sampling to the right side in steps C and D.*

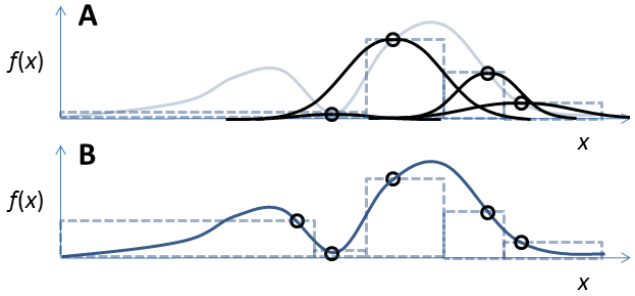

Figure 6: *Treating the kD-tree as a mixture of Gaussians. The mixture components are shown as the black curves in step A. The Gaussians are centered at the samples, with standard deviations proportional to the hypercube widths in each dimension. This blurs the distribution model adaptively, with less blurring where samples are densely concentrated, and increases the chance of samples crossing valleys, illustrated in step B.*

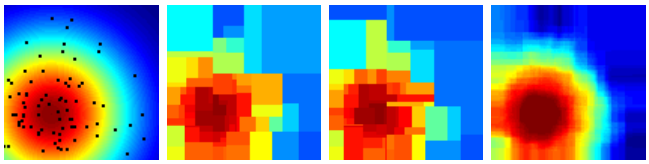

Figure 7: *Random trees. Left: a function and random samples. Middle: two different kD-trees built from the samples on the left. Right: average of 100 random trees.*

with each sample.

The basic idea is to store the sample coordinates and objective function values $\{\mathbf{x}_i, f(\mathbf{x}_i)\}$ into a kD-tree built over the domain. The tree adaptively subdivides the parameter space into hypercubes. Each leaf node is a hypercube, and the volume V_i of each leaf gives an approximate measure of the local density of samples. The weight

$$w_i = f(\mathbf{x}_i)V_i \tag{1}$$

gives a single-sample estimate of the integral of $f(\mathbf{x})$ over the leaf hypercube. One can interpret the tree as a piecewise constant approximation of $f(\mathbf{x})$, from which one may draw samples by first randomly selecting a hypercube with the selection probabilities $\propto w_i$, and then generating a sample uniformly inside the hypercube. However, as shown in Figure 5, this naïve kD-tree sampling easily leads to biases if the value of $f(\mathbf{x})$ evaluated at a sample is not representative of the whole hypercube.

The solution is to treat the kD-tree as a mixture of Gaussians, illustrated in Figure 6. For the selected hypercube i, the sample is drawn from $\mathcal{N}(\mathbf{x}_i; \mathbf{C}_i)$. The covariance $\mathbf{C}_i$ is diagonal with elements $c_{jj} = (\sigma d_{ij})^2$, where σ is a scaling parameter (we use $\sigma = 0.5$) and d_{ij} is the width of the leaf hypercube i along dimension j. The Gaussian tails overlap the neighboring hypercubes, which makes it more likely for the sampling to cross valleys and recover from the biases.

Until a sampling budget is met, we draw a new sample from the mixture, add the sample to the tree, split the leaf node where the new sample lands between the new and old samples, and recompute the weights for the two new leaves (Equation 1).

3.4 The Sequential kD-tree Sampler

In our application, the fitness landscape varies from frame to frame as the environment changes. The phenomenon is illustrated in Figure 8, where changing the throw target changes the objective. To support changing landscapes, we now construct a Sequential Monte Carlo sampler (Algorithm 2) based on the adaptive sampler described above. Building a sampling distribution out of the samples taken during the previous frame allows us to exploit temporal coherence.

At frame t_j, the set of samples from the previous frame t_{j-1} is first pruned to $M < N$ samples by retaining the M samples whose leaf nodes have the largest weights (lines 2-5). A large M means that old samples get selected and re-evaluated often, which may affect convergence in rapidly changing situations, whereas a low value makes tracking multiple modes difficult as little information is retained between frames. We use $M = 0.1N$ in all our results.

After pruning, the tree is rebuilt by inserting the remaining M samples in random order (lines 6-10). This is crucial to avoid persistent spatial biases in the sampling, as the tree building order affects the hypercube volumes and, consequently, the sample weights. However, the randomization averages the biases out temporally. Figure 7 shows a function with samples, three different kD-tree models built from the same set of samples, and an average of 100 trees. We have also experimented with building an ensemble of trees for each time step to further reduce the variance of the volume estimates for each sample, but we do not so far have conclusive evidence of the benefits.

After rebuilding we introduce new samples drawn from a set of heuristics (lines 11-15, see Section 4.5). After this, the sampling prior is complete: the tree contains the M best samples from the

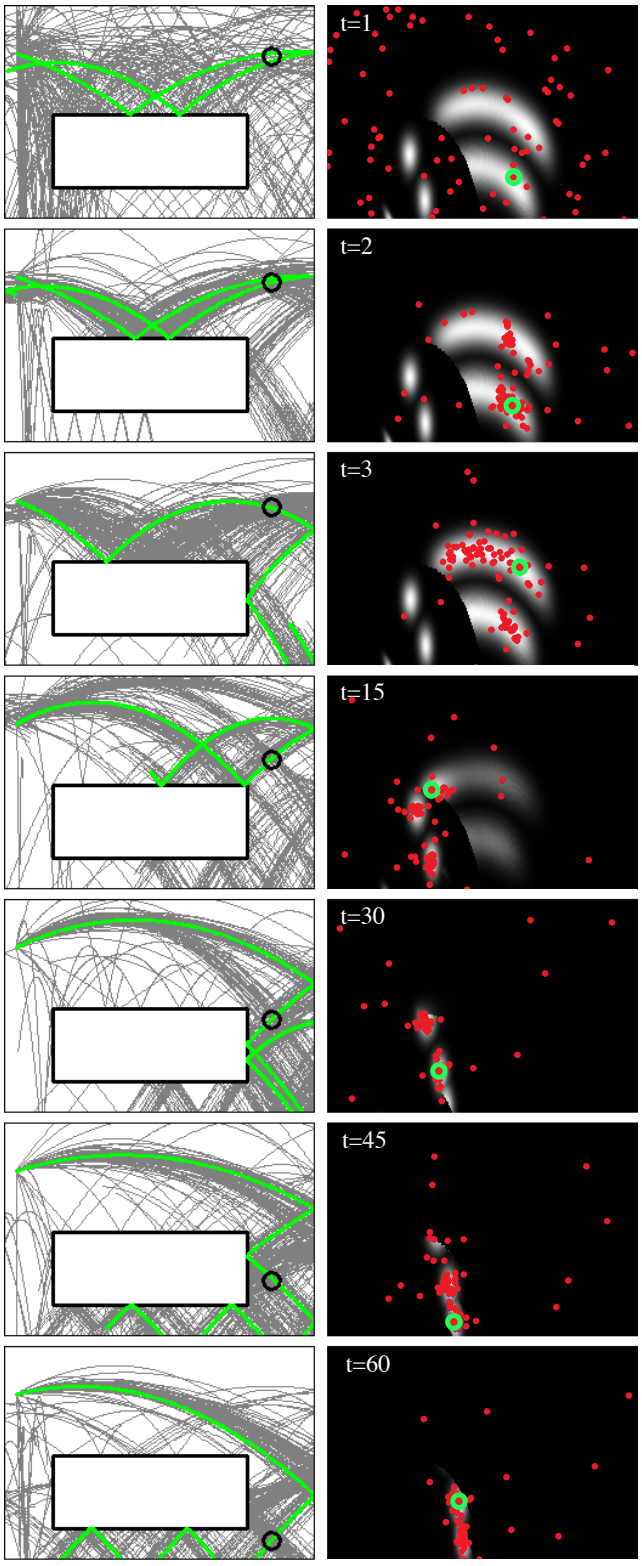

Figure 8: *Tracking of the objective function landscape with 100 samples as the ball throw target moves. The red circles and gray trajectories show the samples generated at each time step, and the green trajectory and green circle show the selected best sample. The benefit of multimodality is evident, as old modes vanish and new ones appear. In the first frame (t=1), the samples are initialized uniformly.*

Algorithm 2 kD-Tree Sequential Importance Sampling

1: **for** each time step t_j **do**
   // Prune tree to M samples
2: **while** #samples $> M$ **do**
3: find leaf i with minimum w_i
4: REMOVETREE($\mathbf{x}_i$)
5: **end while**
   // Randomly shuffle and rebuild tree using old fitnesses
6: CLEARTREE()
7: $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \leftarrow$ RANDOMPERMUTE($\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$)
8: **for** $i = 1 \ldots M$ **do**
9: INSERTTREE($\mathbf{x}_i$)
10: **end for**
   // Draw guesses from heuristics and ML predictors
11: **for** $i = 1 \ldots K$ **do**
12: $\mathbf{x}_g \leftarrow$ DRAWGUESS()
13: evaluate $f(\mathbf{x}_g; t_j)$
14: INSERTTREE($\mathbf{x}_g$)
15: **end for**
16: $\{w_1, \ldots, w_{M+K}\} \leftarrow$ UPDATELEAFWEIGHTS()
   // Then, perform adaptive sampling
17: **repeat**
18: Randomly select leaf node i with probability $\propto w_i$
19: **if** node contains old fitness $f(\mathbf{x}_i; t_{j-1})$ **then**
20: compute current fitness $f(\mathbf{x}_i; t_j)$
21: $w_i \leftarrow V_i f(\mathbf{x}_i; t_j)$ ▷ update weight
22: **else** // Sample as in Algorithm 1
23: draw a sample $\mathbf{x}_{new} \sim \mathcal{N}(\mathbf{x}_i, \mathbf{C}_i)$
24: Evaluate $f(\mathbf{x}_{new}; t_j)$
25: $\{n_1, n_2\} \leftarrow$ INSERTTREE($\mathbf{x}_{new}$)
26: $w_{n_1} \leftarrow V_{n_1} f(\mathbf{x}_{new}; t_j)$
27: $w_{n_2} \leftarrow V_{n_2} f(\mathbf{x}_{n_2}; t_j)$ ▷ $f(\mathbf{x}_{n_2}; t_j)$ known
28: **end if**
29: **until** #samples $= N$
30: **end for**

previous frame, along with new samples generated by heuristics. The remainder of the algorithm performs adaptive sampling much like Algorithm 1 (lines 17-29). The only difference is that when a leaf that contains a stale fitness value from the previous frame is selected, it is recomputed and the weight updated, but a new sample is not generated (lines 20-21). When a node with an up-to-date fitness is selected for refinement, sampling proceeds as in Algorithm 1 (lines 23-27). When the budget of N samples is reached, the current sample set is approximately distributed according to the new fitness $f(\mathbf{x}; t_j)$.

Figure 8 shows how Algorithm 2 tracks the objective function modes in the 2D example when the ball throw target is moving.

3.5 Greedy Sampling

After nearly exhausting our sample budget, we further opportunistically explore the region around the current best sample $\mathbf{x}_b$. We modify Algorithm 2 so that for the last N_g samples of a time step, the selection (line 18) always chooses the leaf with the best sample so far, and a lower scaling factor σ_g is used for computing $\mathbf{C}_i$. Adjusting N_g and σ_g allows one to tune the balance between local and global search. We use $\sigma_g = 0.005$. Section 5 presents the results from our character motion synthesis with different values of N_g.

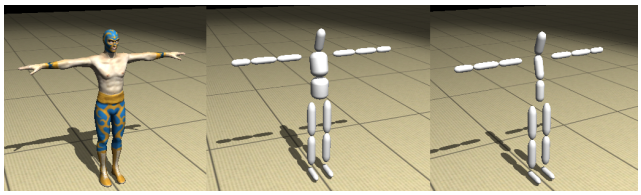

Figure 9: *The character model (left), and heavy and light versions of the physics skeleton used for simulation.*

4 System Description

4.1 Character Model

Figure 9 shows our character and physics models. In our tests, we use two physics models: one light-boned, and another with a considerably heavier torso and thus a higher center of mass (COM), which makes balancing and acrobatics more difficult. The physics objects have constant densities. The character has 30 actuated DOF and 6 unactuated root DOF. The physics model consists of 15 bones (rigid bodies) connected using 3-DOF ball and 1-DOF hinge joints, the latter used for elbows, knees and ankles. We do not model clavicles, toes, and fingers.

For simulation, we use Open Dynamics Engine (ODE) 0.12, using ODE's direct "big matrix" LCP solver, a time step of $\Delta t = 1/30$ seconds, and CFM and ERP parameters as 10^{-5} and 0.2, respectively. Note that ODE also has an iterative solver, which is faster but less stable. For approximately similar quality, the iterative solver requires a timestep of 1/120s, which results in slower operation. In our case, the direct solver only takes approximately as much CPU time as collision handling.

4.2 Optimization Parameterization

We represent control strategies as time-varying target joint angles that are encoded as a sequence of control points of an interpolating cubic spline. We use $n = 4$ control points in all our experiments. Our spline is non-uniform, i.e., the positions of the control points along the temporal axis (the knots) are subject to optimization. Specifically,

$$\mathbf{x} = [\mathbf{z}_1, ..., \mathbf{z}_n], \quad \text{with } \mathbf{z}_i = [\mathbf{q}_i, \mathbf{l}_i, t_i], \tag{2}$$

where the $\mathbf{q}_i$ denote the 30 target joint angles at time t_i. The time coordinate is expressed as an offset from the previous control point, or from the current time for the first control point. The $\mathbf{l}_i$ are limits on the maximum allowable torques for the actuated joints; allowing them to vary instead of using fixed maximums allows the character, e.g., to soften landings from high jumps. We use the sampling bounds $50Nm < \mathbf{l}_i < 150Nm$ for the lightweight model and $50Nm < \mathbf{l}_i < 200Nm$ for the heavier one. The torque limits are specified for three groups of bones: torso, arms and legs. The non-uniform knot sequence allows fine-grained control of fast movements such as jumps.

The total number of optimized variables is 136 (i.e. $\mathbf{x} \in \mathbb{R}^{136}$), consisting of 30 target angles, 3 torque limits, and 1 time coordinate for each of the 4 control points.

4.3 Evaluating the Strategy by Forward Simulation

The spline defined by $\mathbf{x}$ gives continuous-time target joint angles $\mathbf{q}(t; \mathbf{x})$ and limits on maximum torque $\mathbf{l}(t; \mathbf{x})$. To evaluate the objective function, we feed these targets into the physics simulation and record what happens.

The total duration of the simulation for each sample varies depending on the control points. However, we only run the simulation up to a predefined planning horizon. Section 5 presents results with different planning horizons. The default used in the supplementary video is 2 seconds.

For each time step t_j, we evaluate the target angles $\mathbf{q}(t_j; \mathbf{x})$ and torque limits $\mathbf{l}(t_j; \mathbf{x})$ and feed them to the ODE physics simulator. ODE motors are controlled by a target velocities that ODE tries to make the motors reach. We compute the target velocity for the ith motor from the target pose as $(q_i^{\text{target}} - q_i^{\text{current}})/\Delta t$ with q_i denoting joint angles as seen by the motor. The simulator is then stepped forward by the time step; internally, it drives the motors to try to match the target velocities, while respecting the torque limits.

The simulation produces a time series of realized 3D positions $\mathbf{b}$ and velocities $\dot{\mathbf{b}}$ for all the 15 rigid bodies in the skeleton, representing what happened when we tried controlling the character using the strategy $\mathbf{x}$. We denote the series by $\mathbf{S}(\mathbf{x}) = \{\mathbf{b}(t_j; \mathbf{x}), \dot{\mathbf{b}}(t_j; \mathbf{x})\}_{j=-1}^{N_s(\mathbf{x})}$, where $N_s(\mathbf{x})$ is the number of simulated time steps for the sample and $j = 1$ denotes the current time step. We also use the shorthand notation $\mathbf{b}^{(j)} = \mathbf{b}(t_j; \mathbf{x})$. Given the sequence, the objective function then measures the fitness value of the realization using formulae given in Section 4.4.

The time index j of $\mathbf{S}(\mathbf{x})$ starts from -1, as we also include a history of past two frames in the evaluation, which allows the objective function evaluation to prefer continuity with previously selected control strategies. This reduces movement jitter, which could otherwise be a problem in a stochastic sampling system like ours.

Note that as the LCP solver will adapt the torque based on, e.g., contact forces, our scheme provides a slightly higher level of control than using PD controllers. The human motor system comprises both motion-inducing and stabilizer muscles, and stabilizing the character in a given pose is easier with the motors than with PD controllers, especially with the large Δt we use.

Appendix A describes important implementation details related to obtaining a causal, reproducible simulation.

4.4 Objective Function

The objective function drives the character towards the desired goals. In this paper, our goal is to balance the character upright in a predetermined "ready" stance defined by a pose vector $\mathbf{q}_r$, shown in the first frame of Figure 1. The corresponding objective function is formulated as

$$f(\mathbf{S}) = f_d f_s \max[f_b, w_u f_u] \tag{3}$$

where f_d, f_s, f_b, f_u denote damage avoidance, smoothness, balancing, and get up objectives, respectively, and w_u adjusts the priority of the get up objective. We use $w_u = 0.0001$. All components of the objective function are functions of $\mathbf{S}$, but in the following, we omit the dependence for brevity.

Damage avoidance The damage avoidance objective tries to avoid high-velocity impacts to important body parts. We include head and pelvis in the objective to allow versatile movement but prevent the character from falling on its head or bottom. The objective is formulated as

$$f_d = \begin{cases} 1 & \text{if } n_c = 0 \\ \max_i g(|\mathbf{v}_i \cdot \mathbf{n}_i|), \quad i = [1, ..., n_c] & \text{if } n_c > 0 \end{cases} \tag{4}$$

where n_c is the number of all important body contacts during the realized plan and $\mathbf{v}_i$ and $\mathbf{n}_i$ are the relative velocity and contact

normal of the ith contact, respectively. The function g is a soft threshold function $g(x) = 0.5 + 0.5 \tanh[c(t_d - x)]$, where t_d is the threshold and c is a steepness parameter. We use $t_d = 2.2$ and $c = 2.5$.

Smoothness The smoothness objective consist of minimizing acceleration and jerk (the time-derivative of acceleration), which has been found to help in producing natural movement [Van Welbergen et al. 2010]. The objective is given by

$$f_s = e^{-\frac{1}{2}\left(\frac{\mu_a}{\sigma_a^2} + \frac{\mu_J}{\sigma_j^2}\right)}, \tag{5}$$

where μ_a and μ_J are the mean squared acceleration and jerk, respectively, computed as:

$$\mu_a = \frac{1}{N_s(\mathbf{x}) - 1} \sum_{j=1}^{N_s(\mathbf{x})-1} \|\ddot{\mathbf{b}}^{(j)}\|^2,$$
$$\mu_J = \frac{1}{N_s(\mathbf{x})} \sum_{j=0}^{N_s(\mathbf{x})-1} \|\dddot{\mathbf{b}}^{(j)}\|^2, \quad \text{with} \tag{6}$$
$$\ddot{\mathbf{b}}^{(j)} = \frac{\dot{\mathbf{b}}^{(j)} - \dot{\mathbf{b}}^{(j+1)}}{\Delta t}, \; \dddot{\mathbf{b}}^{(j)} = \frac{\dot{\mathbf{b}}^{(j-1)} - 2\dot{\mathbf{b}}^{(j)} + \dot{\mathbf{b}}^{(j+1)}}{(\Delta t)^2}.$$

We use values $\sigma_a = 5.0$ and $\sigma_j = 13.7$. The jerk term μ_J is affected by the history of the last two frames ($j = -1, j = 0$) to avoid acceleration jitter from frame to frame.

Balancing The balancing objective is the most complex one, comprising a desired target pose, desired up-vector direction, velocity minimization, COM displacement minimization and penalization based on other body parts than feet touching the ground. We define it as

$$f_b = \max_j f_g^{(j)} e^{-\frac{1}{2}||\mathbf{r}_b^{(j)}||^2} \tag{7}$$

where j denotes the jth frame. Here, f_g is the penalizing term that gets the value 0 if any body part other than the feet is touching the ground, and 1 otherwise. The vector $\mathbf{r}_b$ is defined as

$$\mathbf{r}_b^{(j)} = \left[\frac{\dot{\mathbf{c}}^{(j)\,T}}{\sigma_{vel1}}, \frac{\dot{\mathbf{b}}^{(j)\,T}}{\sigma_{vel2}}, \frac{\mathbf{d}_{com}^{(j)\,T}}{\sigma_{disp}}, \frac{\mathbf{d}_{up}^{(j)\,T}}{\sigma_{up}}, \frac{\mathbf{q}_d^{(j)\,T}}{\sigma_{pose}} \right]^T \tag{8}$$

Velocity minimization is done by the following terms: $\dot{\mathbf{c}}^{(j)}$ is the velocity of the center of mass, projected to the ground plane. $\dot{\mathbf{b}}^{(j)}$ is the concatenation of the velocities of all bodies.

The COM displacement $\mathbf{d}_{com}^{(j)}$ is computed relative to the COM of the target balancing pose in a character-centric coordinate system defined by the global up vector, and the character facing direction vector projected to the ground plane, with the origin at midway between the feet. Similarly, the up vector difference $\mathbf{d}_{up}^{(j)}$ is computed in the character centric coordinates as $\mathbf{d}_{up} = \mathbf{u}_{root} - \mathbf{u}_{balanced}$, where $\mathbf{u}_{root}$ is the up-vector of the root bone, and $\mathbf{u}_{balanced}$ is the corresponding vector of the target balancing pose.

Finally, $\mathbf{q}_d$ denotes the differences between the local joint angles of the simulated and target poses, using angles as seen by ODE motors. The values we use for the scaling multipliers are $\sigma_{vel1} = 0.05m/s$, $\sigma_{vel2} = 0.75m/s$, $\sigma_{disp} = 0.05m$, $\sigma_{up} = 0.1m$ and $\sigma_{pose} = 15.0$ degrees.

The balancing objective is computed for each frame between a specified minimum time and the length of the planning horizon, and the best scoring frame is used for evaluating the whole sample. We use 0.5s as the minimum. In many MPC systems, the objective function is partitioned into a running cost and a terminal cost, and the terminal cost is only evaluated at a fixed planning horizon. In contrast, our system allows the optimizer some slack in terms of when to reach the goals, which should make the objective function modes larger and thus easier to find and track.

Get-Up The get up objective is the same as the balancing objective, but omitting the pose term and having less strict velocity multipliers $\sigma_{vel1} = 1.0m/s$ and $\sigma_{vel2} = 15.0m/s$.

$$f_u = \max_j f_g^{(j)} e^{-\frac{1}{2}||\mathbf{r}_u^{(j)}||^2}, \tag{9}$$

where j is the jth frame and

$$\mathbf{r}_u^{(j)} = \left[\frac{\dot{\mathbf{c}}^{(j)\,T}}{\sigma_{vel1}}, \frac{\dot{\mathbf{b}}^{(j)\,T}}{\sigma_{vel2}}, \frac{\mathbf{d}_{com}^{(j)\,T}}{\sigma_{disp}}, \frac{\mathbf{d}_{up}^{(j)\,T}}{\sigma_{up}} \right]^T, \tag{10}$$

where terms are computed similar to f_b, but the COM displacement term also includes the y-component clamped to $y = min(0, y - (y_{target} + h))$, where $h = 0.1m$ denotes an offset of the COM that is higher than in the target balancing pose. This is to not penalize motion where the COM is temporarily above the balancing pose, e.g., when taking steps or leaping.

The main difference to previous work is that the objective function formulation is multimodal due to the $\max$ function. In practice, we have found that when the character has fallen down, the sampler has difficulties maximizing all the components of the balancing objective f_b. However, maximizing f_u is much easier, and very often leads to a situation where, as a consequence, f_b is easier to maximize. In effect, this is similar to how Jain et al. [2009] define balancing as a sequence of states including balancing and taking a step. However, we do not need to define explicit state transitions or target locations for foot placement, and the sampler may freely pick the best strategy in each situation. The f_u component also allows the character to take steps to regain balance or dodge projectiles, as it does not penalize deviations from the target pose.

Roll With the heavier model in the middle of Figure 9, even a get up strategy is sometimes difficult to find when the character has fallen on its back. In these cases, we add a third alternative objective f_r inside the $\max$ function in Equation 3 that makes the character roll away from its back

$$f_r = \max_j w_r e^{-20||y_f^{(j)}+1||^2} \tag{11}$$

where y_f is the y-coordinate of the character's normalized forward vector computed from the torso, and w_r is the priority of the rolling. We use a small $w_r = 10^{-40}$ because we want the character to keep improvising alternative get up strategies if possible.

4.5 Heuristics and Machine Learning

Heuristics At each frame, we generate 20% of the samples uniformly within the parameter space. We also add a guess where each control point of the spline equals the target balancing pose and where joint torque limits are constant. Finally, we add the best sample $\mathbf{x}_b$ of the previous frame after stepping its parameters one time step forward, i.e., shifting the spline backward in time by Δt.

When evaluating the last heuristic, it is important to ensure that the interpolated results from the original $\mathbf{x}_b$ and the shifted spline match to machine precision over the entire horizon. We achieve this by writing the spline evaluator recursively, such that splines are only ever evaluated at $t = 0$, and stepping forward by Δt is handled by changing the knots and control points.

Machine Learning Our system supports the optional generation of guesses (lines 11-15 in Algorithm 2) from an arbitrary machine learning component, with the idea of drawing on previous experience to infer good strategies for the current situation. We use approximate nearest neighbors query using the FLANN library [Muja and Lowe 2009] to map a feature vector to a set of control strategies that are injected as guesses on line 12. The feature vectors consist of current pose angles, the "up" direction of the root node, the rotation and velocity of the root node, and optionally, the relative position and velocity of the closest projectile to dodge. The training set is normalized with respect to L_2 norms of the feature variables.

We train the mapping during online optimization (the ball evading test explained in Section 5), storing the feature vector and best sample $\mathbf{x}$ for all frames where $f(\mathbf{x}) > 10^{-10}$.

While the simple ANN search is probably not the optimal machine learning method for our case, Section 5.2 shows that as few as 3 approximate nearest neighbors improve get-up performance considerably. We consider our implementation a proof-of-concept of integrating machine learning with the SMC sampling of control strategies; development and evaluation of more efficient and expressive learning components is left as future work.

5 Results

We have tested our method in three ways: 1) throwing spheres at the character, 2) adding sudden impulses to body parts to disturb balance and throw the character around, and 3) triggering simulated explosions that add impulses to all body parts. Figures 1, 10, 11, and 14 illustrate these tests.

In the tests, the character is able to avoid the spheres — the avoidance behavior implicitly caused by the jerk minimization goal — recover lost balance in creative ways, such as rolling over the shoulders to land back on its feet, and get up when thrown to the ground. We describe the results both qualitatively (Section 5.1) and quantitatively (Section 5.2).

In the following, we refer to the supplemental video using the time in parenthesis (mm:ss).

Performance The supplemental video was captured in real-time (using Fraps, `www.fraps.com`) on a Windows 7 PC with Intel Core i7-4930k 3.40GHz CPU (12 logical cores), and an NVIDIA GeForce GTX 480 GPU. On this computer, the optimizer runs at approximately 20 fps with a 1/30s physics time step, $N = 25$ samples per frame, and a planning horizon of 2 seconds. On a 2012 MacBook Pro laptop with a 2.4GHz processor, the same settings yield 6-10 fps, enough for interactive experimenting and parameter tuning, which we consider one of the best aspects of the system. As shown in the video, 25 samples is enough to synthesize a variety of movements, whereas using 100 samples (01:39) slows the simulation down considerably. On the other hand, using fewer samples per frame or a shorter planning horizon yields fully real-time but unreliable results (01:21).

5.1 Qualitative evaluation

The system shows considerable creativity in adapting to surprising situations and utilizing the environment. For example, the character dodges the spheres using pirouette jumps (02:22) and slides to dodge a rolling sphere, using a hand to keep the sphere away (00:32). When the character's head is punched to the ground, it continues the movement as a cartwheel of sorts and rises up (00:49). Taking steps emerges as an avoidance strategy (02:42, Figure 11), although not always successfully (01:48). The character also often lands on its feet when thrown in the air (00:00, 00:38).

The top left corner of the video shows which of the alternative objective function components gives the highest score for the best scoring sample. "Balancing" corresponds to f_b and "Getting up" to $w_u f_u$. Using the 2s planning horizon, the sampler is often able to find a balancing strategy while still rolling on the ground after an impact (01:02, 01:12).

The main drawbacks of the system are that movement is sometimes stiff and has unnecessary joint contortions (02:18). The stiffness is probably caused by our parameterization using target angles instead of joint torques. The torque limit optimization does help, e.g., in softening landings; however, the sampling and/or the goals are not able to relax the character's hands in many cases. The character also often keeps the hands greedily close to the target pose even when not nearly balanced. We experimented with shoulder and elbow torque minimization goals, but this easily leads to the other extreme of the hands hanging limp, which does not look natural for our fighter character. The heuristic balancing initial guess can also cause the character to assume the target pose prematurely while still moving (02:03). Sometimes this appears almost unphysical, as the character uncannily knows that although it is swaying, it will ultimately end up balanced. Without the heuristic or machine learning guesses, however, the character keeps hovering about the target pose, illustrating a typically slow final convergence of global sampling methods. Combining global sampling with local refinement is clearly a topic for future work.

In the future, one easy way to improve the naturalness of the movements could be scripted or learned control of gaze and head orientation. For example, real humans typically follow flying objects with their gaze, and try to look at the expected landing spot while airborne. The hand and foot contacts with the ground could also be fine-tuned, e.g., so that the character always exerts forces on the ground using the palm instead of fingertips. We expect that this can be done using purely visual corrections based on inverse kinematics instead of modifying the optimization, but this remains future work.

We have also tested two other balancing poses - an asymmetric Taido (a martial art) ready stance and a regular standing position. Both poses work, although the regular standing appears more difficult - it is less stable as the support polygon is smaller and COM is higher.

5.2 Quantitative evaluation

The system is stochastic, and hence may occasionally provide good results even with just a few samples. To ensure that our results are representative, we have run a quantitative balancing and avoidance test with varying parameters. In each test, 100 spheres are thrown at the character from random directions. The spheres are 3x heavier than the character, i.e., failure to avoid the ball almost certainly leads to the character falling down. We measured the percentage of times the character was balanced 5 seconds after the ball was thrown, determined by thresholding the objective function value.

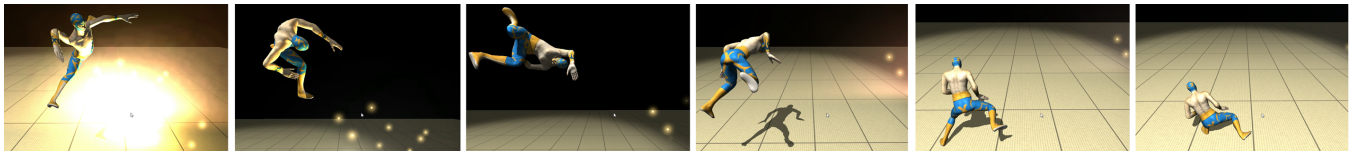

Figure 10: *The user triggers an explosion and the character flips around in the air to land on its feet.*

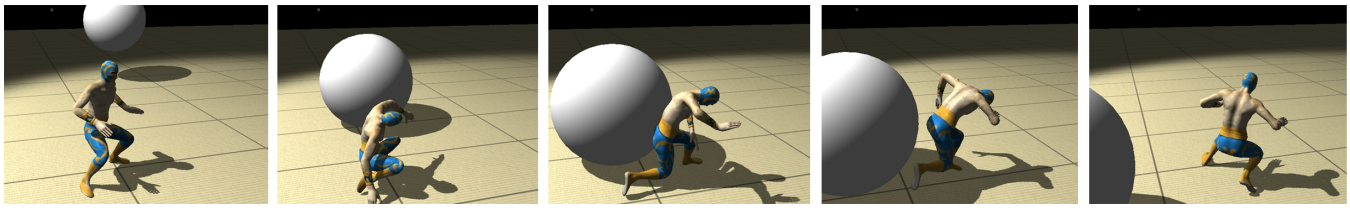

Figure 11: *Footsteps emerge as the character dodges the balls thrown at it.*

To succeed, the character could either dodge the ball, or get successfully up after a failed dodge. The test also saves a screenshot of each failure case. The most typical cases are wide splits and lying on the back. The supplementary video shows that these are difficult situations (01:33, 01:48).

The left side of Figure 12 shows the success percentage as a function of optimizer samples per frame in four conditions. ST denotes the "standard" setup used in capturing the supplemental video (2s planning horizon, lightweight character model). In ST+ML, 3 FLANN predictions were generated in each frame from a dataset of 100k training vectors, which yielded better results at low sample budgets. This indicates that our system can utilize machine learning as intended. The HV curve denotes the heavier character model with no changes compared to ST, which yields abysmal success rates at low sample budgets. Performance is better in the HV2 case, where we activated the "roll away from back" goal, used a 3.5s planning horizon, and measured success after a longer period of 8 seconds after each ball throw.

The right side of Figure 12 shows the successful attempts as a function of the greedy sampling parameter N_g. There appears to be a sweet spot of 25-50% greedy samples. All our tests and the supplemental video capturing use $N_g = 25\%$.

Figure 13 shows the successful attempts as a function of the number of samples and the length of the planning horizon. One can see that the 2s horizon used in the supplementary video is a reasonable default, and longer horizons do not produce considerable benefit.

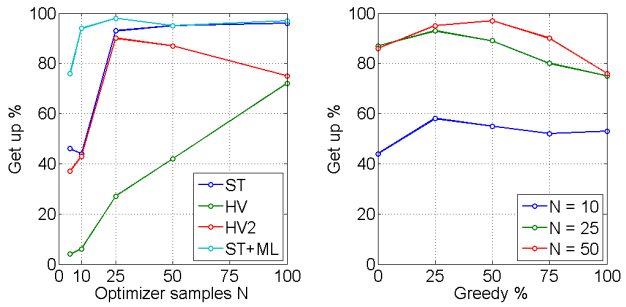

Figure 12: *Get up percentage as a function of samples per frame (left) and get up percentage as a function of greedy sampling percent (right) in a test where 100 heavy balls were thrown at the character.*

6 Conclusion

We have demonstrated that Sequential Monte Carlo (SMC) sampling is a viable approach for online synthesis of complex human movements without reliance on animation or motion capture data. The central features of the system are the use of kD-trees for sampling, non-uniform splines for pose interpolation, and a rigid body physics engine with custom modifications to ensure reproducible simulations. While the key component, an adaptive sequential sampling method, allows easy integration of machine learning to draw on previous experience, we are surprised by the performance of the sampler even without machine learning or using dimensionality reduction methods to constrain the search space.

We have integrated our system with Unity3D, a state-of-the-art commercial game engine. The results will be released as open source. However, we believe our sampler is simple enough to also implement from scratch.

We see improving performance and controlling the style of synthesized movement as the two main items for future work. We expect that both can be addressed by precomputing a suitable prior for the sampling, and/or developing an interactive training application where the user may instruct a machine learning system to learn the most interesting movements that have emerged. Our parameterization also allows for pose-space dimensionality reduction, and according to our initial experiments, it does make abnormal poses less frequent. However, heavy dimensionality reduction using a small training set easily overconstrains the movement while a larger training set allows the character to use poses in abnormal contexts, e.g., kicking while balancing. Contextual and temporal information could be incorporated, e.g., by using offline optimization to generate a training set of control splines that follow motion capture trajectories, similar to [Muico et al. 2009].

In the future, we also plan to explore novel interactions and game mechanics utilizing the motion synthesis, and investigate whether sequential sampling is competitive also in offline synthesis, where the function landscape changes over time when the animator interactively adjusts parameters. It could also be interesting to simulate muscle animation, breathing, grunting etc. based on the predicted near-future exertion (e.g., "I'll jump and breathe out in one second, better breath in now").

Figure 14: *The user gives an impact to the left upper arm, causing the character to fall on its back. The emerging getting up strategy comprises first rolling over to the right side, and then using the right hand as a support to allow moving the right foot so that weight can be shifted on it.*

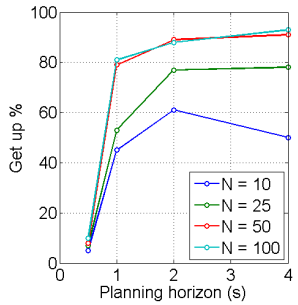

Figure 13: *Get up percentage as a function of samples per frame (N) and the planning horizon.*

Acknowledgments

We thank all the reviewers for their valuable comments. The research has been supported by the Skene - Games Refueled program of the Finnish Funding Agency for Innovation.

A Implementation Notes

We ended up using ODE and its direct LCP solver, as the iterative solvers in ODE, Bullet Physics or PhysX engines were not stable enough for active characters except at very small time steps that were not computationally efficient. The 3-DOF hip and shoulder joints were especially unstable, and although previous studies have successfully used 2-DOF joints [Tassa et al. 2012], 3-DOF joints are needed for a realistic character with a skinned mesh. It appears that the current mainstream physics engines are optimized for passive objects and ragdolls, although a new version of Bullet has just appeared with new solvers geared towards robotics.

We have made two important changes to ODE. Firstly, the original implementation of ODE is not causal due to some internal implementation details such as reordering of arrays for optimization purposes and due to the way random numbers are generated. We have solved these issues by removing the non-deterministic optimizations and by storing the random number generator seed on a threading context level. This ensures that running two simulations in different threads with the same control parameters achieve exactly the same motion. If the simulation is not fully causal, the sampler sometimes forgets a chosen control strategy before it has been completely executed.

Secondly, ODE implements joint motor limits in a way that might cause too much force to be applied when the motor is moving away from the limit. This causes instability, and ODE has solved this by introducing a hand-tuned fudge factor that scales the force. Getting the maximum available force and the fudge factor correct for each body part is delicate and difficult, and to solve this we used a fudge-free patch from the official ODE issue tracker that instead adds the motor limits as constraint rows in the LCP formulation. This makes the simulation more robust.

Our threading uses a pool of worker threads, which each obtain a sample from the sampler, simulate the physics forward, compute $f(\mathbf{S})$ and store the computed value to the sampler. Access to the sampler is synchronized, which means that our implementation is not optimal for massively parallel computing. However, with our current computers with up to 12 logical cores, we have achieved a decent 75-80% core utilization.

References

AL BORNO, M., DE LASA, M., AND HERTZMANN, A. 2013. Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics 19*, 8, 1405–1414.

ARULAMPALAM, M., MASKELL, S., GORDON, N., AND CLAPP, T. 2002. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing 50*, 2, 174–188.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Proc. SIGGRAPH '92*, ACM, New York, NY, USA, 293–302.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Simulation of human motion data using short-horizon model-predictive control. *Computer Graphics Forum 27*, 2, 371–380.

DA SILVA, M., DURAND, F., AND POPOVIĆ, J. 2009. Linear bellman combination for control of character animation. In *Proc. SIGGRAPH 2009*, ACM, New York, NY, USA, 82:1–82:10.

DE VILLIERS, J. P., GODSILL, S. J., AND SINGH, S. S. 2011. Particle predictive control. *Journal of Statistical Planning and Inference 141*, 5 (May), 1753–1763.

DEUTSCHER, J., BLAKE, A., AND REID, I. 2000. Articulated body motion capture by annealed particle filtering. In *IEEE Conference on Computer Vision and Pattern Recognition, 2000. Proceedings*, vol. 2, 126–133 vol.2.

DOUCET, A., AND JOHANSEN, A. M. 2009. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering 12*, 656–704.

EREZ, T., LOWREY, K., TASSA, Y., KUMAR, V., KOLEV, S., AND TODOROV, E. 2013. An integrated system for real-time model-predictive control of humanoid robots. In *Proc. IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*, HUMANOIDS.

FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Trans. Graph. 22*, 3, 417–426.

GEIJTENBEEK, T., PRONOST, N., EGGES, A., AND OVERMARS, M. H. 2011. Interactive character animation using simulated physics. *Eurographics-State of the Art Reports 2*.

GEIJTENBEEK, T., VAN DE PANNE, M., AND VAN DER STAPPEN, A. F. 2013. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph. 32*, 6 (Nov.), 206:1–206:11.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '07, 129–136.

HÄMÄLÄINEN, P., AILA, T., TAKALA, T., AND ALANDER, J. 2006. Mutated kd-tree importance sampling. In *Proc. SCAI 2006*, 39–45.

IHLER, A. T., SUDDERTH, E. B., FREEMAN, W. T., AND WILLSKY, A. S. 2003. Efficient multiscale sampling from products of gaussian mixtures. *Advances in Neural Information Processing Systems 16*, 1–8.

JAIN, S., YE, Y., AND LIU, C. K. 2009. Optimization-based interactive motion synthesis. *ACM Trans. Graph. 28*, 1 (Feb.), 10:1–10:12.

KAJIYA, J. T. 1986. The rendering equation. In *Proc. SIGGRAPH '86*, ACM, New York, NY, USA, 143–150.

KANTAS, N., MACIEJOWSKI, J. M., AND LECCHINI-VISINTINI, A. 2009. Sequential monte carlo for model predictive control. In *Nonlinear Model Predictive Control*, L. Magni, D. M. Raimondo, and F. Allgöwer, Eds., no. 384 in Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, Jan., 263–273.

LASSETER, J. 1987. Principles of traditional animation applied to 3D computer animation. In *Proc. SIGGRAPH '87*, ACM, New York, NY, USA, 35–44.

LIU, L., YIN, K., VAN DE PANNE, M., SHAO, T., AND XU, W. 2010. Sampling-based contact-rich motion control. *ACM Trans. Graph. 29*, 4, 128:1–128:10.

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

MUICO, U., LEE, Y., POPOVIĆ, J., AND POPOVIĆ, Z. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph. 28*, 3, 81:1–81:9.

MUJA, M., AND LOWE, D. G. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISAPP (1)*, 331–340.

NGO, J. T., AND MARKS, J. 1993. Spacetime constraints revisited. In *Proc. SIGGRAPH '93*, ACM, New York, NY, USA, 343–350.

PEJSA, T., AND PANDZIC, I. 2010. State of the art in example-based motion synthesis for virtual characters in interactive applications. *Computer Graphics Forum 29*, 1, 202–226.

REIL, T., AND HUSBANDS, P. 2002. Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation 6*, 2, 159–168.

RUDOY, D., AND WOLFE, P. 2006. Monte carlo methods for multi-modal distributions. In *Proc. Fortieth Asilomar Conference on Signals, Systems and Computers, 2006. ACSSC '06*, 2019–2023.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph. 23*, 3, 514–521.

SCHMIDT, J., FRITSCH, J., AND KWOLEK, B. 2006. Kernel particle filter for real-time 3D body tracking in monocular color images. In *Proc. 7th International Conference on Automatic Face and Gesture Recognition, 2006. FGR 2006*, 567–572.

SIMS, K. 1994. Evolving virtual creatures. In *Proc. SIGGRAPH '94*, ACM, New York, NY, USA, 15–22.

STAHL, D., AND HAUTH, J. 2011. PF-MPC: particle filter-model predictive control. *Syst Control Lett 60*, 8, 632–643.

TASSA, Y., EREZ, T., AND TODOROV, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proc. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, IROS'12, 4906–4913.

THRUN, S., FOX, D., AND BURGARD, W. 2000. Monte carlo localization with mixture proposal distribution. In *Proc. AAAI/IAAI*, 859–865.

VAN WELBERGEN, H., VAN BASTEN, B. J. H., EGGES, A., RUTTKAY, Z. M., AND OVERMARS, M. H. 2010. Real time animation of virtual humans: A trade-off between naturalness and control. *Computer Graphics Forum 29*, 8, 2530–2554.

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

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Proc. SIGGRAPH '88*, ACM, New York, NY, USA, 159–168.