To appear in ACM Transactions on Graphics (Proc. SIGGRAPH 2015)

Online Control of Simulated Humanoids Using Particle Belief Propagation

Perttu Hämäläinen[1*] Joose Rajamäki[1†] C. Karen Liu[2‡]

[1] Aalto University [2] Georgia Tech

Figure 1: *Our algorithm can handle complex balancing and manipulation tasks while adapting to user interactions. All our demonstrated movements emerge from simple cost functions without animation data or offline precomputation. More examples can be found in the supplemental video and on the project homepage.*

Abstract

We present a novel, general-purpose Model-Predictive Control (MPC) algorithm that we call Control Particle Belief Propagation (C-PBP). C-PBP combines multimodal, gradient-free sampling and a Markov Random Field factorization to effectively perform simultaneous path finding and smoothing in high-dimensional spaces. We demonstrate the method in online synthesis of interactive and physically valid humanoid movements, including balancing, recovery from both small and extreme disturbances, reaching, balancing on a ball, juggling a ball, and fully steerable locomotion in an environment with obstacles. Such a large repertoire of movements has not been demonstrated before at interactive frame rates, especially considering that all our movement emerges from simple cost functions. Furthermore, we abstain from using any precomputation to train a control policy offline, reference data such as motion capture clips, or state machines that break the movements down into more manageable subtasks. Operating under these conditions enables rapid and convenient iteration when designing the cost functions.

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

Keywords: animation, motion synthesis, motion planning, optimization

*e-mail:perttu.hamalainen@aalto.fi

†e-mail:joose.rajamaki@aalto.fi

‡e-mail:karenliu@cc.gatech.edu

1 Introduction

Research on procedural, physically based humanoid movement synthesis has attracted considerable attention, due to its promise of transforming the animator or game designer into a choreographer that directs the characters using high-level commands instead of laboriously editing animation frame-by-frame. As shown by the seminal work of Witkin and Kass [1988], physically based motion synthesis can be formulated as an optimization problem, yielding plausible movement with desirable qualities such as "squash-and-stretch" and anticipation. The field has since progressed from simple characters with a few rigid bodies towards humanoid models of increasing biomechanical detail, from offline to real-time simulation, and from purely optimization-based synthesis to utilizing libraries of motion capture and animation data. Detailed reviews of the state-of-the-art can be found in [Geijtenbeek and Pronost 2012; Guo et al. 2014; Pejsa and Pandzic 2010]. Although complex movements like bipedal locomotion can now be simulated in real-time, challenges remain in making the characters autonomously adapt to unpredictable environments, such as obstacles to foot placement and sudden changes in direction when steered interactively.

In this paper, we set us the challenge of avoiding the following approaches that most physically based real-time systems rely on: 1) using pre-scripted or recorded data such as motion capture clips, 2) employing offline precomputation to learn a control policy, or 3) designing specialized state machines that break down movement into more manageable parts, e.g., defining the recovery of balance as first lifting a foot, and then placing the foot down in a prescripted direction. All the approaches require significant offline effort and/or result in less robust real-time behavior in novel situations. Instead, we utilize the power of our online optimization algorithm to make adaptive real-time behavior emerge from simple cost functions.

Our work utilizes the fact that current personal computers can simulate rigid body dynamics hundreds of times faster than real-time. This has presented an opportunity for a new class of Model-Predictive Control (MPC) methods that use black-box dynamics simulators such as Open Dynamics Engine to simulate a controlled system forward at each control update up to a planning horizon of a few seconds. The approach has been gaining popularity in both animation and robotics research, and has been demonstrated to work with simulated humanoids and the requirements above

[Hämäläinen et al. 2014; Tassa et al. 2014], although only a limited variety of behavior has been demonstrated so far.

In this paper, we introduce a novel forward-simulating MPC method based on Particle Belief Propagation, a generic sampling-based belief propagation method [Ihler and Mcallester 2009]. Compared to the state-of-the-art of local optimization [Tassa et al. 2012; Tassa et al. 2014; Toussaint 2009], our method is robust to cost function discontinuities and multimodality. On the other hand, related sampling methods employ some form of particle filtering (a.k.a. Sequential Monte Carlo) [Hämäläinen et al. 2014; Stahl and Hauth 2011], which has in some other domains been superseded by belief propagation [Sudderth et al. 2010]. Sampling-based belief propagation allows effective factorization of high-dimensional problems, and has not been previously applied to the control of simulated humanoids. A detailed comparison to previous work is given in Section 3.

We demonstrate our method in near real-time (20-30fps) control of a simulated humanoid character with 30 actuated degrees of freedom (DOF) and 6 unactuated root DOFs. As shown in Figure 1 and on the supplemental video, our system is capable of a wide range of robust and adaptive behavior. For example, the character automatically takes steps and shuffles feet to reach a target, places a hand on a wall or ground to recover balance, and catches a ball with a different limb if being disturbed while juggling. Our work advances the state-of-the-art considering 1) the variety of the adaptive behavior, and 2) the demonstration of novel, complex behavior such as balancing and reaching on top of a ball, and interactive, steerable locomotion with the capability of climbing over obstacles. The main limitations are that our sampling method produces some movement noise, and we use a simulation model with simple joint actuators instead of more detailed muscle and tendon models. Qualitywise, we do not aim to compete with state-of-the-art offline or data-based methods such as [Wampler et al. 2014; Mordatch et al. 2012; Geijtenbeek et al. 2013]. Similar to [Hämäläinen et al. 2014], our contribution is more in interactivity, i.e., in allowing a control designer to rapidly test a wide range of objective functions and interactions without time-consuming data gathering or offline computation.

2 Method

We first give a brief overview of our method, illustrated in Figure 2. Our algorithm works by doing path finding and smoothing in time, and then transmitting between animation or game frames the information that is generated in the process. This comprises the following main phases:

- Explore the time evolution of the system forward up to a planning horizon K using N guided random walkers. In other words, we sample control vectors and use dynamics simulation to obtain the corresponding next states for N trajectories. This corresponds to Figure 2A.
- Smooth the optimal control obtained in the previous phase by recursive backwards local refinement. This is shown in Figure 2B.
- Deploy the obtained control to the system and transmit information of the hypothesized time evolution to the next frame or iteration. The result of the algorithm performed with this prior data is seen in Figure 2C.

A cost function is evaluated for each forward simulation step and each trajectory. If many of the trajectories have a high total evaluated cost, we perform a resampling step, marked with vertical blue lines in Figure 2A. The resampling redistributes the computing resources by pruning the less likely (high cost) trajectories and forking the most likely ones.

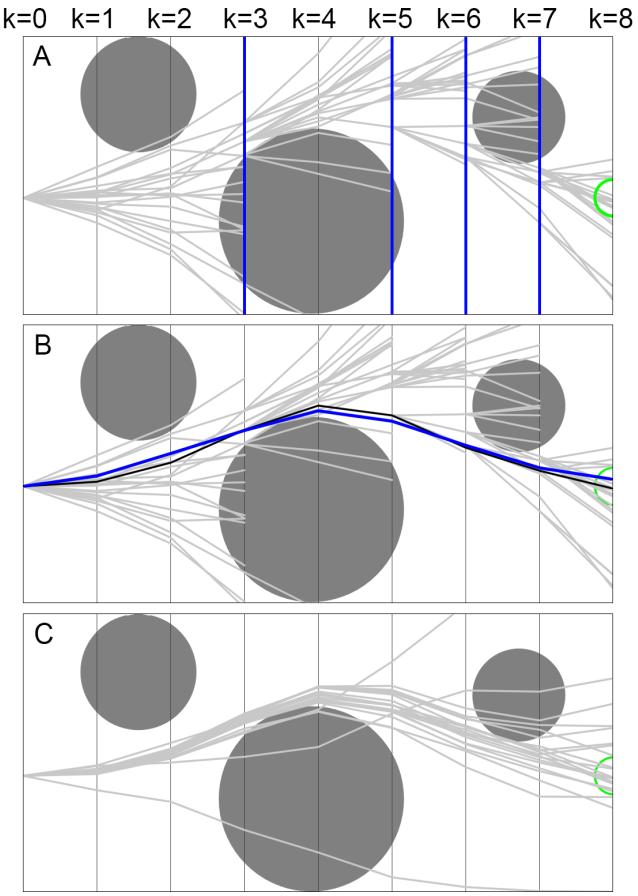

Figure 2: *Our method illustrated in a case where the controlled object moves from left to right at a constant velocity (i.e., k denotes both time and horizontal position), vertical acceleration is controlled, and the green circle denotes the target. Figure A shows the trajectories sampled during the forward pass. The vertical blue lines indicate time instants of resampling. In Figure B the black line indicates the optimal trajectory found in the forward pass and the blue line indicates the results of a backwards local refinement pass. Figure C shows the trajectories sampled by our method in the next iteration, using the previous trajectories as a prior.*

The forward and backward passes constitute a single iteration of the algorithm. The trajectories of the previous iteration are used to guide the random walkers of the next iteration. In online control, we run one iteration per frame, assuming that the simulated system exhibits only small changes between frames.

2.1 Notational conventions

Let us denote by n the frame of an animation or a game. Inside a frame n, we can hypothesize about the time evolution of the system forward with different controls and system dynamics. We denote the time instant difference from the given frame by index k. Now, we can denote by $a_{k,n}$ a quantity a at time $k+n$ as seen from the frame n. Thus, $k < 0$ refers to the past time. The time instants $k > 0$ are hypotheses about the future and $k = 0$ refers to the given frame.

If the quantity a has only one lower index, e.g. a_k, it is used to

refer to the time difference k from the "current" frame. If we have a group of samples of quantity a indexed by i, we will denote them by $a^{(i)}$. As an example, $a_{3,1}^{(2)}$ denotes the sample number 2 of a at time instant $4 = 3 + 1$ as seen in frame 1. In a context where the frame information is irrelevant we could simply use $a_3^{(2)}$. As samples may represent values of continuous trajectories, we also use a notation $a_k^{(h(i,k,-\delta))}$, where $h(i, k, -\delta) = j$ is a history function that maps the sample index i at k to the corresponding sample index j of the same trajectory at $k - \delta$. This slightly cumbersome notation is needed as the resampling operations remap the indices between steps.

2.2 Online control

In many control systems the system model is of the form:

$$\mathbf{x}_t = f_t(\mathbf{x}_{t-1}, \mathbf{u}_t) + \boldsymbol{\xi} \tag{1}$$

$$\boldsymbol{\xi} \sim \mathcal{N}(0, \mathbf{Q}). \tag{2}$$

In this discrete time model $\mathbf{x}_t$ represents the system state and $\mathbf{u}_t$ represents a control signal at time t. The state transition function f_t tells how the system evolves at time t. The process has zero mean Gaussian noise that is denoted by $\boldsymbol{\xi}$. $\mathbf{Q}$ stands for the noise's covariance matrix.

To make any claims about the optimality of a control-policy one has to define cost for the system state and control. Let us denote the state and control dependent cost at time instant t by $\ell_t(\mathbf{x}_t, \mathbf{u}_t)$. In this article we assume the cost to be a positive valued function and to contain separate state and control cost terms:

$$\ell_t(\mathbf{x}_t, \mathbf{u}_t) = s_t(\mathbf{x}_t) + c_t(\mathbf{u}_t). \tag{3}$$

For an arbitrary cost function and arbitrary system dynamics, the computation of the optimal control that minimizes cost is often intractable. The complex, changing contacts of simulated humanoids in particular make the transition function highly nonlinear and discontinuous, and a closed-form expression for the transition function may not be available. In the following, we limit ourselves to evaluating $f_t(\mathbf{x}_{t-1}, \mathbf{u}_t)$ pointwise by running a forward dynamics simulation step with starting state $\mathbf{x}_{t-1}$ and control $\mathbf{u}_t$. We use Open Dynamics Engine as our simulator.

Throughout sections 2.3-2.5, we will make use of the toy example of Figure 2 to explain our method. In the toy example, $\mathbf{u}_t$ is scalar and simply denotes the vertical acceleration at time t. The state $\mathbf{x}_t \in \mathbb{R}^2$ includes vertical position and velocity. Collisions and gravity are disabled, and collisions only have effect through $s_t(\mathbf{x}_t)$ which is proportional to squared obstacle penetration depth and vertical distance from target. The control cost favors small accelerations, $c_t(\mathbf{u}_t) \propto \|\mathbf{u}_t\|^2$. Our 3D biped is more complex and operates under collisions and gravity, as explained in Section 4.

2.3 Control as Markov Random Field

Let $\mathbf{z}_k = [\mathbf{x}_k \, \mathbf{u}_k]^T$ denote the combined state and control vector at forward prediction step k. We turn the cost minimization into finding the maximum of the probability density $\mathcal{P}(\mathbf{z})$ of the full trajectories $\mathbf{z} = [\mathbf{z}_0, ..., \mathbf{z}_K]$, where K denotes the planning horizon. This is approached through 1) transforming cost functions into probability densities through exponentiation (e.g., [Todorov 2008]), and 2) approximate sampling from $\mathcal{P}(\mathbf{z})$, which is likely to produce samples at the density peaks.

If one only considers the state and control costs, one can define $\mathcal{P}(\mathbf{z})$ as

$$\begin{aligned}\mathcal{P}(\mathbf{z}) &= \frac{1}{Z}\prod_k \exp\Big[-\frac{1}{2}\big(s_k(\mathbf{x}_k) + c_k(\mathbf{u}_k)\big)\Big] \\ &= \prod_k \alpha_k(\mathbf{x}_k)\beta_k(\mathbf{u}_k) \\ &= \prod_k \psi_k(\mathbf{z}_k),\end{aligned} \tag{4}$$

where Z is a normalization term, and α_k and β_k denote state and control *potential* functions that have their maxima at zero costs. $\psi_k(\mathbf{z}_k)$ is a shorthand for the combined state and control potential. The potentials are Gaussian if the costs are quadratic, but as explained in section 5, for a simulated biped the state cost may depend on contacts and can thus be discontinuous and multimodal. An example of such discontinuity is a damage cost for the character hitting the ground with its head.

Equation (4) implicitly assumes that the $\mathbf{z}_k$ always form a valid trajectory in the combined state and control space. However, considering the $\mathbf{z}_k$ as separate random variables, we must augment the equation with the probabilities of each $\mathbf{z}_k$ actually connecting with the adjacent $\mathbf{z}_{k-1}$ and $\mathbf{z}_{k+1}$:

$$\begin{aligned}\mathcal{P}(\mathbf{z}) &= \frac{1}{Z}\Big(\prod_k \psi_k(\mathbf{z}_k)\Big) \\ &\quad \Big(\prod_{k=1}^{K} \Psi_{\text{fwd}}(\mathbf{z}_{k-1}, \mathbf{z}_k)\Big)\Big(\prod_{k=0}^{K-1} \Psi_{\text{bwd}}(\mathbf{z}_{k+1}, \mathbf{z}_k)\Big),\end{aligned} \tag{5}$$

where Ψ_{fwd} and Ψ_{bwd} denote forward and backward *transition potentials*, defined using Equation (1) as

$$\begin{aligned}\Psi_{\text{fwd}}(\mathbf{z}_{k-1}, \mathbf{z}_k) &= \Psi_{\text{bwd}}(\mathbf{z}_k, \mathbf{z}_{k-1}) \\ &= \mathcal{N}(\mathbf{x}_k; f_k(\mathbf{x}_{k-1}, \mathbf{u}_k), \mathbf{Q}),\end{aligned} \tag{6}$$

where $\mathcal{N}(\mathbf{x}; \mu, \mathbf{C}) = \exp[-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{C}^{-1}(\mathbf{x}-\mu)]$, i.e., an unnormalized Gaussian function of $\mathbf{x}$ with mean μ and covariance matrix $\mathbf{C}$. We omit the usual Gaussian normalization terms as they can be included in the Z, and later on we need to evaluate the potentials also in the limit when $\mathbf{Q} = 0$. Now, $\lim_{\mathbf{Q}\to 0} \Psi_{\text{fwd}}(\mathbf{z}_{k-1}, \mathbf{z}_k) = 1, \mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \mathbf{u}_k)$, and zero otherwise.

Equation (5) corresponds to a simple case of a Markov Random Field (MRF), where the variables $\mathbf{z}_k$ are conditionally independent of others than the neighboring $\mathbf{z}_{k-1}, \mathbf{z}_{k+1}$. This can also be depicted as the probabilistic graphical model of Figure 3. Each node of the model corresponds to the $\mathbf{z}_k$ of one forward prediction step, and the lines denote the dependencies between the variables. The graphical model formulation forms the basis for the rest of the paper, as it allows us to apply belief propagation (BP) methods.

In more general terms, an MRF model can be written as [Ihler and Mcallester 2009]:

$$\mathcal{P}(\mathbf{z}) = \frac{1}{Z}\Big(\prod_s \psi_s(\mathbf{z}_s)\Big)\Big(\prod_{\{s,t\}\in E} \Psi_{s,t}(\mathbf{z}_s, \mathbf{z}_t)\Big), \tag{7}$$

where the indices s and t denote source and target nodes of a graphical model. E is the set of edges in the model.

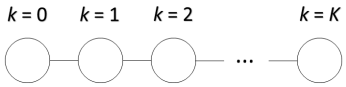

Figure 3: *Our graphical model, with each node representing the state and control variables for a forward prediction step, up to the planning horizon K.*

2.4 Particle Belief Propagation

Building on the MRF formulation above, we now examine how the toy example of Figure 2 can be mapped to Particle Belief Propagation (PBP), a general-purpose sampling-based belief propagation method [Ihler and Mcallester 2009]. The motivation for using PBP is that it combines two powerful approaches: global sampling for handling multimodal optimization landscapes, and dynamic programming to fight the curse of dimensionality. Here, we utilize PBP to be able to sample the marginals $\mathcal{P}_k(\mathbf{z_k})$ instead of attempting to directly sample the much higher-dimensional $\mathcal{P}(\mathbf{z})$. In other words, we sample and operate on trajectory segments instead of full trajectories. In the end, we are not interested in the marginal $\mathcal{P}_k(\mathbf{z}_k)$ as such, but estimating them through belief propagation will allow us to also produce samples of good full trajectories, as explained in detail in Section 2.5.

Similar to other belief propagation approaches, PBP seeks to compute the beliefs $\mathcal{B}_k(\mathbf{z}_k)$ of the variables of a graphical model. The beliefs are proportional to the marginal $\mathcal{P}_k(\mathbf{z}_k)$ if the graphical model is a simple chain or tree. The belief for node k depends on messages $m_{s\to k}(\mathbf{z}_k)$ passed from other nodes [Ihler and Mcallester 2009]:

$$\mathcal{B}_k(\mathbf{z}_k) = \psi_k(\mathbf{z}_k) \prod_{s\in\Gamma_k} m_{s\to k}(\mathbf{z}_k), \tag{8}$$

$$m_{s\to k}(\mathbf{z}_k) = \sum_{\mathbf{z}_s\in Z_s} \Psi_{s,k}(\mathbf{z}_s,\mathbf{z}_k)\psi_s(\mathbf{z}_s) \prod_{u\in\Gamma_s\setminus k} m_{u\to s}(\mathbf{z}_s). \tag{9}$$

Here, Γ_s denotes the set of neighbors of node s, and Z_s is the domain of $\mathbf{z}_s$. The messages $m_{s\to k}$ from node s to node k can be considered as (unnormalized) probability density functions of the target node variables $\mathbf{z}_k$. The potentials $\psi_k(\mathbf{z}_k)$ represent the "evidence" for $\mathbf{z}_k$ (here based on the state and control costs), which propagate through the graphical model via the messages. The belief $\mathcal{B}_k(\mathbf{z}_k)$ equals the product of the direct and propagated evidence.

In Particle Belief Propagation, the messages and beliefs of equations (8) and (9) are estimated by samples $\mathbf{z}_k^{(i)} \sim q_k(\mathbf{z}_k^{(i)})$, where i denotes the sample index and $q_k(\mathbf{z}_k^{(i)})$ is an arbitrary proposal distribution. Division by the proposal then gives the importance-weighted sample messages and beliefs [Ihler and Mcallester 2009]:

$$\widehat{m}_{s\to k}(\mathbf{z}_k^{(i)}) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{s,k}(\mathbf{z}_s^{(j)},\mathbf{z}_k^{(i)}) \frac{\psi_s(\mathbf{z}_s^{(j)})}{q_s(\mathbf{z}_s^{(j)})} \prod_{u\in\Gamma_s\setminus k} \widehat{m}_{u\to s}(\mathbf{z}_s^{(j)}), \tag{10}$$

$$\widehat{\mathcal{B}}_k(\mathbf{z}_k^{(i)}) = \frac{\psi_k(\mathbf{z}_k^{(i)})}{q_k(\mathbf{z}_k^{(i)})} \prod_{s\in\Gamma_k} \widehat{m}_{s\to k}(\mathbf{z}_k^{(i)}) \tag{11}$$

where N denotes the number of samples.

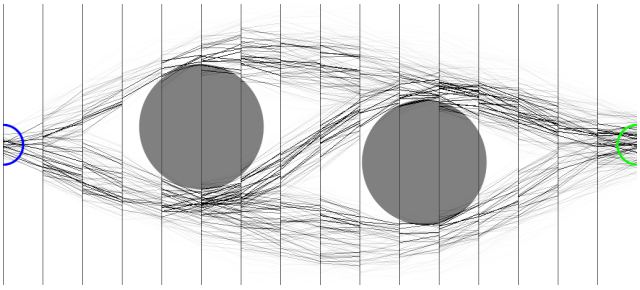

Figure 4: *A direct application of the Particle Belief Propagation algorithm to a similar control problem as in Figure 2.*

In our case the sample belief $\widehat{\mathcal{B}}_k(\mathbf{z}_k^{(i)})$ represents the marginal probability of a trajectory segment produced by a simulation step with controls $\mathbf{u}_k^{(i)}$ ending at state $\mathbf{x}_k^{(i)}$. Although this does not yet yield us an optimal control or state trajectory, we can already visualize the beliefs as shown in Figure 4. The figure was obtained by drawing $\mathbf{z}_k^{(i)}$ from a simple uniform $q_k(\mathbf{z}_k^{(i)})$ except for $\mathbf{x}_0^{(i)}$ which is fixed, computing the sample beliefs according to Equation (11), and plotting the segments corresponding to each $\mathbf{z}_k^{(i)}$ with higher beliefs mapped to darker colors[1]. Note that the covariance matrix $\mathbf{Q}$ acts as a tuning parameter; with a too low transition noise, the low transition potentials between segments can make the beliefs evaluate to zero within machine precision. A too high transition noise, on the other hand, blurs the marginals and makes the peaks less distinct.

Figure 4 shows that basic PBP is able to approximate the multimodal marginal distributions corresponding to different ways to pass the obstacles. The correct information for the control problem is clearly there, but has yet to be recovered, which brings us to our method.

2.5 Our Method

Based on the generic PBP algorithm described above, our algorithm makes several novel contributions for controlling complex simulated characters, including 1) a choice of proposal that enables the sampling of physically feasible trajectories, 2) adaptive resampling to adjust between local and global search, 3) a local refinement backward pass, 4) avoiding zero message values, 5) propagating information not only within the graph in Figure 3 but also between animation frames, 6) including various control priors to smooth movement and aid convergence, and 7) handling the special case of $\mathbf{Q} = 0$. We call the resulting method Control Particle Belief Propagation (C-PBP), summarized in Algorithm 2.

The most glaring problem of the basic PBP example of Figure 4 is that the sampled trajectory segments do not connect, i.e., they cannot be directly combined to form physically feasible trajectories. The sampled starting position of a segment can also lie inside an obstacle. Thus, the following starts with the choice of proposal that yields feasible, realizable trajectories for which one can evaluate $\mathcal{P}(\mathbf{z})$. The best sampled trajectory can then be used both as the deployed control and a warm-start for the next frame. The beliefs are used to inform sampling in subsequent iterations (frames).

The Proposal Density Following standard importance sampling practice, the proposal should be as close to the target density as

[1]For plotting, we need both an initial and final state for each segment. Fortunately, the toy example dynamics are linear due to no collisions, and we can compute the initial state based on the sampled $\mathbf{z}_k^{(i)}$.

possible. We implement this by having a cloud of random walkers that start from the known initial state, and whose controls are drawn following the control potential, $\mathbf{u}_k^{(i)} \sim \beta_k(\mathbf{u}_k^{(i)})$. The corresponding states are computed using the dynamics simulator, i.e., $\mathbf{x}_k^{(i)} = f(\mathbf{x}_{k-1}^{(h(i,k,-1))}, \mathbf{u}_k^{(i)})$. Here, we use the $h(\dot{)}$ notation as the sample indices may be remapped between forward prediction steps due to resampling.

As the drawn $\mathbf{u}_k^{(i)}$ fully determine $\mathbf{x}_k^{(i)}$, $q(\mathbf{z}_k^{(i)})$ depends only on $\mathbf{u}_k^{(i)}$, and we may write

$$q(\mathbf{z}_k^{(i)}) = \beta_k(\mathbf{u}_k^{(i)}). \tag{12}$$

The chosen proposal quarantees the connectivity of trajectory segments and simplifies the sample beliefs and messages of Equations (10) and (11) so that they only depend on the state potentials. Here, we also utilize the fact that our graphical model is a simple chain, which allows us to divide the messages into forward and backward messages $\widehat{m}_{\text{fwd}}$ and $\widehat{m}_{\text{bwd}}$:

$$\widehat{\mathcal{B}}_k(\mathbf{z}_k^{(i)}) = \alpha_k(\mathbf{x}_k^{(i)})\widehat{m}_{\text{fwd}}(\mathbf{z}_k^{(i)})\widehat{m}_{\text{bwd}}(\mathbf{z}_k^{(i)}), \tag{13}$$

$$\widehat{m}_{\text{fwd}}(\mathbf{z}_k^{(i)}) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)}, \mathbf{z}_k^{(i)})\alpha_{k-1}(\mathbf{x}_{k-1}^{(j)}) \widehat{m}_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)}), \tag{14}$$

$$\widehat{m}_{\text{bwd}}(\mathbf{z}_k^{(i)}) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{\text{bwd}}(\mathbf{z}_{k+1}^{(j)}, \mathbf{z}_k^{(i)})\alpha_{k+1}(\mathbf{x}_{k+1}^{(j)}) \widehat{m}_{\text{bwd}}(\mathbf{z}_{k+1}^{(j)}). \tag{15}$$

When implementing the algorithm, the forward messages can be updated in a single forward pass over the graph, followed by a backward pass that updates the backward messages and beliefs.

Resampling During the forward pass, one can express the forward sample messages recursively in terms of forward beliefs

$$\widehat{B}_{\text{fwd}}(\mathbf{z}_k^{(i)}) = \alpha_k(\mathbf{x}_k^{(i)})\widehat{m}_{\text{fwd}}(\mathbf{z}_k^{(i)}), \tag{16}$$

$$\widehat{m}_{\text{fwd}}(\mathbf{z}_k^{(i)}) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)}, \mathbf{z}_k^{(i)})\widehat{B}_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)}). \tag{17}$$

$\widehat{B}_{\text{fwd}}(\mathbf{z}_k^{(i)})$ is proportional to the full belief without information passed backwards along the graph. In the cases where the backward messages do not add much information (e.g., no obstacles ahead in the toy example), one may improve convergence by greedily terminating paths with low forward belief and increasing the exploration around the paths with high forward belief. We implement this as standard particle filter resampling (e.g., [Arulampalam et al. 2002]), noting that the forward belief is analogous to sample weights in particle filtering. At the beginning of time step k, we compute the effective particle count of the previous step

$$N_{\text{eff}} = \frac{1}{\sum_i \left(\frac{\widehat{B}_{\text{fwd}}(\mathbf{z}_{k-1}^{(i)})}{\sum_j \widehat{B}_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)})}\right)^2}, \tag{18}$$

where the denominator equals the sum of squared normalized beliefs. A small N_{eff} indicates that the beliefs of all but a few trajectories are close to zero. If $N_{\text{eff}} < T_{\text{eff}}N$, we first create a probability distribution function for sample index $\mathcal{P}(r)$ such that $\mathcal{P}(r) \propto \widehat{B}_{\text{fwd}}(\mathbf{z}_{k-1}^{(r)})$. For each sample i, we randomly draw a sample index r from $\mathcal{P}(r)$ and assign the history function such that the sample i is now mapped to sample r in the previous time step. This terminates low belief trajectories and forks high belief ones, as shown in Figure 2A. Similar to particle filtering, where sample weights are set to constant after resampling, we omit the $\widehat{B}_{\text{fwd}}(\mathbf{z}_{k-1}^{(j)})$ from the forward message of Equation (17) at steps where resampling takes place.

In the simulations of this paper, $T_{\text{eff}} = 0.5$. $T_{\text{eff}} < 1/N$ corresponds to no resampling, in which case the trajectories are simulated up to the planning horizon even if they pass right through an obstacle in the toy example. $T_{\text{eff}} > 1$ corresponds to resampling in every frame, which may result in premature termination of good trajectories.

As illustrated in Figure 2A, sampling the marginal $\mathbf{z}_k$ with our proposal and resampling results in a tree of trajectories. Backtracking the trajectories that reach the planning horizon $k = K$ allows one to recover N full control and state trajectories. The full density $\mathcal{P}(\mathbf{z})$ can now be evaluated for the trajectories, and the best trajectory used as a result, or a local refinement of the trajectories can be attempted, as explained in the following.

Local Refinement Since the dynamics simulation is computationally heavy, it makes sense to try to extract as much information as possible from the generated samples. This is why we have added the additional local refinement backward pass given by Algorithm 1, executed after all random walkers reach the planning horizon K. The local refinement helps in the final convergence, decreasing the control cost, which in the case of minimized control forces corresponds to smoothing the trajectory.

Algorithm 1 The local refinement backwards pass.

1: Initialize $\widehat{\mathbf{x}}_K = \frac{\sum_i \alpha(\mathbf{x}_K^{(i)})\mathbf{x}_K^{(i)}}{\sum_i \alpha(\mathbf{x}_K^{(i)})}$
2: **for** $k = K-1 \ldots 1$ **do**
3: Compute a Gaussian model of the concatenated vectors $[\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}, \mathbf{x}_{k+1}^{(i)}]$, weighted by $\widehat{B}_{\text{fwd}}(\mathbf{z}_k^{(i)})$
4: Set $\widehat{\mathbf{x}}_k, \widehat{\mathbf{u}}_k = \text{E}[\mathbf{x}_k, \mathbf{u}_k | \mathbf{x}_{k+1} = \widehat{\mathbf{x}}_{k+1}]$ based on the model.
5: **end for**

At line 1, $\widehat{\mathbf{x}}_K$ is initialized to the mean of the terminal states of the trajectories, weighted by the corresponding state potentials. This is a simple estimate of the optimal final state. Inside the loop of Algorithm 1, a Gaussian model is constructed from the samples $[\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}, \mathbf{x}_{k+1}^{(i)}]$ generated during the forward pass and weighted by the forward belief $\widehat{B}_{\text{fwd}}(\mathbf{z}_k^{(i)})$ (line 3). The next $\widehat{\mathbf{x}}_k$ and $\widehat{\mathbf{u}}_k$ are then computed through the expectation based on the Gaussian model (line 4). Due to the weighting, the expectation maximizes both forward belief and the probability that simulating forward from $\widehat{\mathbf{x}}_k$ with controls $\widehat{\mathbf{u}}_k$ yields $\widehat{\mathbf{x}}_{k+1}$. As $\mathbf{x}$ may be high-dimensional and there may be only a small number of trajectory samples, the expectation is computed in a regularized form

$$\text{E}[\mathbf{x}_k, \mathbf{u}_k | \mathbf{x}_{k+1} = \widehat{\mathbf{x}}_{k+1}] = \boldsymbol{\mu}_{(\mathbf{x}_k, \mathbf{u}_k)} + \boldsymbol{\Sigma}_{(\mathbf{x}_k, \mathbf{u}_k), \mathbf{x}_{k+1}} \left(\boldsymbol{\Sigma}_{\mathbf{x}_{k+1}, \mathbf{x}_{k+1}} + \lambda \boldsymbol{I}\right)^{-1} \left(\widehat{\mathbf{x}}_{k+1} - \boldsymbol{\mu}_{\mathbf{x}_{k+1}}\right). \tag{19}$$

Here μ_a denotes the mean of random variable a and $\Sigma_{a,b}$ denotes

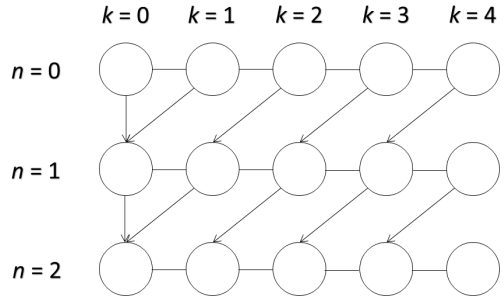

Figure 5: *The graphical model including both animation frames n and forward simulation steps k. The message passing between frames is directed and does not thus cause loops in the model.*

the covariance matrix of random variables a and b. λ is a regularization parameter. We use $\lambda = 0.001$ in all our simulations.

Avoiding Zero Potentials and Messages We have found that in the humanoid simulation, the state potentials can easily evaluate to zero within machine precision in difficult situations with high state costs. This can cause, e.g., the character simply lying immobile on the ground. To remedy this, we scale the state costs so that the smallest state cost for each forward simulation step k does not exceed a threshold $T_\gamma = 20$, which makes the potential functions less peaked:

$$\gamma = \frac{T_\gamma}{\max(T_\gamma, \min_i(s_k(\mathbf{x}_k^{(i)})))} \quad (20)$$

$$\alpha_k(\mathbf{x}_k^{(i)}) = \exp(-0.5\gamma s_k(\mathbf{x}_k^{(i)})). \quad (21)$$

Operation Over Multiple Frames To propagate information over animation frames, we extend our graphical model to include both forward simulation steps and frames, as shown in Figure 5. The figure illustrates how $\mathbf{z}_{k,n}$ is informed by $\mathbf{z}_{k+1,n-1}$, as time progresses between frames.

We also employ the following steps between frames:

- For frame n, we deploy the first control of the best trajectory found in the forward pass of previous frame to define the initial state $\mathbf{x}_{0,n} = f(\mathbf{x}_{0,n-1}, u_{1,n-1}^{(b)})$, where b is the index of the best trajectory. In terms of the graphical model, this means that the message from $\mathbf{z}_{0,n-1}$ to $\mathbf{z}_{0,n}$ is a Dirac delta function in the state dimensions.
- We use the best trajectory as well as the results of Algorithm 1 as warm-starts. The warm-starts occupy two "slots" of the limited sample budget N. The remaining samples are each drawn using samples of the previous frame as a prior, which in the graphical model corresponds to the messages from node $(k+1, n-1)$ to (k, n), as explained below in detail.

Note that we do not deploy the results of Algorithm 1, as the Gaussian models are not always accurate and small errors in estimated controls may lead to large deviations of the resulting state sequence. In case Algorithm 1 produces a good trajectory, it will be adopted as the best trajectory in the next frame.

Control Priors Human movement often conserves energy which is why motion synthesis cost functions typically have terms that penalize high velocity, torque, acceleration, and jerk (the time-derivative of acceleration). We also assume that the environment changes somewhat smoothly between frames. This is why we use the following terms in the control potential and proposal:

$$\begin{aligned}\beta_k(\mathbf{u}_{k,n}^{(i)}) = \; & \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; 0, \sigma_0^2\mathbf{C}_u) \\ & \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; \mathbf{u}_{k-1,n}^{(h(i,k,-1))}, \sigma_1^2\mathbf{C}_u) \\ & \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; 2\mathbf{u}_{k-1,n}^{(h(i,k,-1))} - \mathbf{u}_{k-2,n}^{(h(i,k,-2))}, \sigma_2^2\mathbf{C}_u) \\ & \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; \boldsymbol{\mu}_{k,n}^{(i)}, \mathbf{C}_e) \\ & \mathcal{P}(\mathbf{u}_{k,n}^{(i)} | \mathbf{z}_{:,n-1}^{(:)}), \end{aligned} \quad (22)$$

where we denote by $\mathbf{z}_{:,n-1}^{(:)}$ the possible dependency on more than one forward prediction step and sample of the previous frame.

The first three terms of Equation (22) correspond to minimizing the squared control and its first and second derivatives. $\mathbf{C}_u$ is a diagonal covariance matrix used as a tuning parameter together with the scalars $\sigma_0, \sigma_1, \sigma_2$. For example, if $\mathbf{u}$ is defined as the target angular velocities of joint motors, one may adjust the relative minimization of velocity, acceleration, and jerk using $\sigma_0, \sigma_1, \sigma_2$. $\mathbf{C}_u$ can be set to identity or adjusted to make some joints overall less active.

To clarify the second and third terms of Equation (22), the derivative of the control can be formed using backward finite difference

$$\Delta\mathbf{u}_{k,n} = \frac{\mathbf{u}_{k,n} - \mathbf{u}_{k-1,n}}{\delta t}. \quad (23)$$

Here δt is the time difference between frames or time instants. If we want to keep the derivative zero, we make $\mathbf{u}_{k,n}$ have the highest probability to be $\mathbf{u}_{k-1,n}$ based on Equation (23). Similarly, if we want to set the second derivative equal to zero, we choose $\mathbf{u}_{k,n}$ by the following equation that is obtained by finite differences

$$\Delta^2\mathbf{u}_{k,n} = \frac{\mathbf{u}_{k,n} - 2\mathbf{u}_{k-1,n} + \mathbf{u}_{k-2,n}}{\delta t^2} = 0. \quad (24)$$

This implies that we should make $\mathbf{u}_{k,n}$ have the highest probability to be $2\mathbf{u}_{k-1,n} - \mathbf{u}_{k-2,n}$.

The fourth term in Equation (22) specifies an optional Gaussian prior that can be set by the application. We use this in the bipedal control to specify a pose prior, as explained in Section 4.2.

The last term in Equation (22) represents the information passed from the previous frame. In principle, we could pass a message from node $(k+1, n-1)$ to node (k, n) similar to the backward and forward messages, using the transition potential $\Psi(\mathbf{z}_{k+1,n-1}, \mathbf{z}_{k,n}) = \mathcal{N}(\mathbf{x}_{k-1,n}; \mathbf{x}_{k,n-1}, \mathbf{Q})\mathcal{N}(\mathbf{u}_{k,n}; \mathbf{u}_{k+1,n-1}, \sigma_m^2\mathbf{C}_u)$, which increases exploration around the states and controls that previously had high belief. σ_m is a user-adjusted "mutation" range parameter. However, with a limited sampling budget it is better to use prior information for drawing good samples in the first place instead of discounting the beliefs of samples already drawn. Thus, instead of incorporating the message in Equation (11), we include the corresponding conditional Gaussian Mixture Model (GMM) in Equation (22):

$$\mathcal{P}(\mathbf{u}_{k,n}^{(i)} | \mathbf{z}_{:,n-1}^{(:)}) \propto \sum_j^N w^{(j)} \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; \mathbf{u}_{k+1,n-1}^{(j)}, \sigma_m^2\mathbf{C}_u) \quad (25)$$

$$w^{(j)} = \widehat{\mathcal{B}}_{k+1,n-1}^{(j)}(\mathbf{z}_{k+1,n-1}^{(j)}) \cdot \mathcal{N}(\mathbf{x}_{k-1,n}^{(i)}; \mathbf{x}_{k,n-1}^{(j)}, \mathbf{Q}). \quad (26)$$

Through Equation (26), when drawing the control vector sample $\mathbf{u}_{k,n}^{(i)}$ to simulate forward from a previous state $\mathbf{x}_{k-1,n}^{(i)}$, the weights of the GMM are adjusted based on the beliefs of the previous frame, as well as the distance of the previous state from the trajectories of the previous frame. Note that in order to minimize the number of tuning parameters, we use the same $\mathbf{Q}$ in transitions between frames and between time instants, although Equation (26) could also use a different covariance matrix.

To prevent the sampling from converging to a local optimum, we reserve N_n samples of the sampling budget N to be drawn without using the $\mathcal{P}(\mathbf{u}_{k,n}^{(i)}|\mathbf{z}_{:,n-1}^{(:)})$ term, as noted on lines 21-22 of Algorithm 2.

Allowing Zero Transition Variance Adjusting $\mathbf{Q}$ may be difficult with high-dimensional $\mathbf{x}$, and one might want to first try setting $\mathbf{Q} = 0$. In this case, the transition potentials in equations (14) and (15) are one when j and i belong to the same trajectory, and zero otherwise, which simplifies the message computations as:

$$\widehat{m}_{\text{fwd}}(\mathbf{z}_k^{(i)}) = \alpha_{k-1}(\mathbf{x}_{k-1}^{(h(i,k,-1))})\widehat{m}_{\text{fwd}}(\mathbf{z}_{k-1}^{(h(i,k,-1))}), \quad (27)$$

$$\widehat{m}_{\text{bwd}}(\mathbf{z}_k^{(i)}) = \alpha_{k+1}(\mathbf{x}_{k+1}^{(h(i,k,1))})\widehat{m}_{\text{bwd}}(\mathbf{z}_{k+1}^{(h(i,k,1))}). \quad (28)$$

This reduces the backward message passing to simply backtracking the generated trajectories which provides no additional information. Fortunately, our local refinement backward pass still combines information from multiple trajectories through the Gaussian models (Algorithm 1 Line 3), achieving a similar smoothing effect. As shown in Figure 6, both the local refinement and nonzero $\mathbf{Q}$ improve convergence in the toy problem, and can be used together or separately. The supplemental video illustrates an online case with a moving target, $\mathbf{Q} = 0$ and no local refinement (00:37), $\mathbf{Q} \neq 0$ and no local refinement (00:41), and with both $\mathbf{Q} \neq 0$ and local refinement (00:46). The main difference between the backwards message passing and local refinement (smoothing) is that the former only updates the beliefs, and the latter can yield a trajectory that can lie outside the states and controls of the samples.

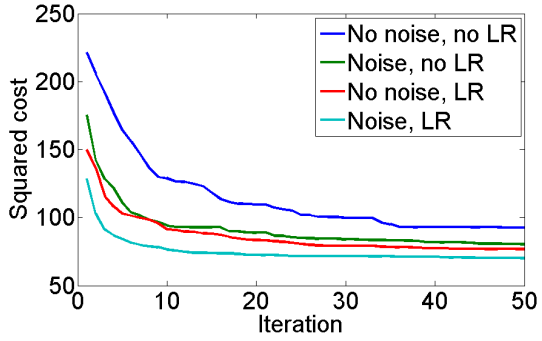

Figure 6: *Convergence of our algorithm in the problem of Figure 2 with zero and non-zero transition noise, and with and without the Gaussian local refinement (LR) pass, plotted as the average of 10 runs. Using both non-zero transition noise and the local refinement produce best results.*

For adjusting $\mathbf{Q}$ one should consider that using equations (27) and (28) instead of (14) and (15) removes the ability of neighboring trajectories to contribute to each other's beliefs through the summations in equations (14) and (15). Figure 7 illustrates how the summation with the Gaussian kernels in Equation (15) is central to the dynamic programming nature of PBP, i.e., the ability to reuse computing results. With $\mathbf{Q} = 0$, the algorithm forgets the trajectories that get pruned in resampling.

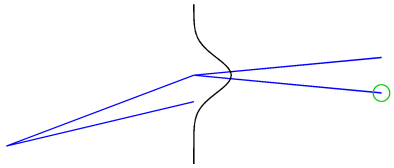

Figure 7: *Even if a trajectory is pruned in the resampling of the forward pass (here the lowest one), its marginal states and controls may gain some belief in the backwards message passing of Equation* (15) *with the Gaussian transition potentials ($\mathbf{Q} \neq 0$) illustrated in black. In this case, the pruned trajectory could yield a smoother path to the target (green circle), and the gained belief will make it influence the sampling of the next frame through Equation* (26)*.*

A modification for $\mathbf{Q} = 0$ is that we do not use the GMM of Equation (25), the weights of which would evaluate to zero for practically all samples. Instead, for each sampled control sequence $\mathbf{u}_{1\ldots K,n}^{(i)}$ we randomly select a previous trajectory $\mathbf{u}_{k=1\ldots K,n-1}^{(j)}$, $\mathcal{P}(j) \propto \widehat{\mathcal{B}}_K^{(j)}$ and use the following in Equation (22) instead of Equation (25):

$$\mathcal{P}(\mathbf{u}_{k,n}^{(i)}|\mathbf{z}_{:,n-1}^{(:)}) = \mathcal{N}(\mathbf{u}_{k,n}^{(i)}; \mathbf{u}_{k+1,n-1}^{(j)}, \sigma_m^2 \mathbf{C}_u). \quad (29)$$

The modified prior of Equation (29) propagates the previous beliefs forward between frames somewhat similar to the GMM of Equation (25). Due to equations (27) and (28), $\widehat{\mathcal{B}}_K^{(j)} = \prod_k \alpha(\mathbf{x}_{k,n-1}^{(j)})$, which is proportional to the importance-weighted probability of the whole trajectory. Using prior control sequences independent of state can be criticized, as the resulting state sequence may deviate in the next frame due to the different initial state. This deviation is fortunately minimal, as due to resampling, the the prior trajectories form a tree with root at $k = 0$, and many of the trajectories initially follow the best trajectory that defines the new initial state. Note that the prior trajectory indices j are propagated in the sampling data structures so that in resampling operations, the forked paths use the prior trajectory of their parent.

Figure 8 illustrates the priors for a single control vector in the case of $\mathbf{Q} = 0$.

3 Related Work

Having described our method and central concepts such as belief propagation, we now consider our work in relation to previous research.

Previous work has demonstrated that robust online motion can be generated in a variety of ways even without the aid of motion capture data. For example, some researchers have shown that good controller design decisions can carry a long way from handling perturbations to adapting new situations [Yin et al. 2007; Coros et al. 2010; Jain et al. 2009; Ha et al. 2012]. Other researchers develop abstract models that capture the essence of dynamics to optimize a longer horizon of motion online [Mordatch et al. 2010; Wu and Popović 2010]. Moreover, leveraging simulated training data to learn control policies has extended reinforcement learning or policy search algorithms to much more challenging applications such as gymnastics or bicycle stunts [Borno et al. 2014; Liu et al. 2012; Tan et al. 2014]. While the independency of motion capture data is a nice feature, some researchers advocate that using a small

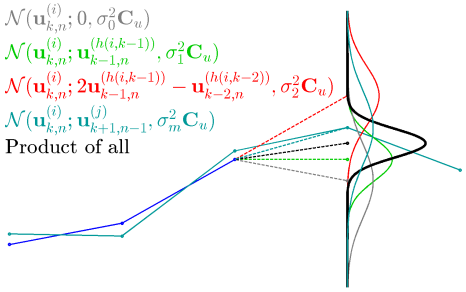

Figure 8: *The sampling distribution for $u_{k,n}^{(i)}$ (See Equation* (22)*). The sampling is done from the black distribution which is the product of the others. The previous values $u_{k-3\cdots k-1,n}^{(i)}$ are drawn as the blue points. Green and red denote the minimization of first and second derivatives of control, respectively. Gray denotes keeping the control at zero. Dark cyan denotes the control in the previous frame according to Equation* (29)*.*

amount of motion capture can greatly enhance the quality of the resulting motion and reduce the burden of parameter tuning [Da Silva et al. 2008; Muico et al. 2009; Ye and Liu 2010; Lee et al. 2010]. All these methods have demonstrated the ability to operate in new or unexpected environments, but their offline dependency prevents them from exploring multimodal control strategies online. For example, none of the previous methods can switch the balance strategy from modulating postures to putting a hand on the wall, without pre-defined state machines, pre-recorded data, or pre-computed policies. In this paper, we developed a model-predictive-control algorithm to make online, multimodal control decisions with minimal offline dependency.

Our work builds on the sampling-based belief propagation methods that have been proven powerful in, e.g., computer vision tracking of articulated bodies [Ihler and Mcallester 2009; Sudderth et al. 2010]. Such methods combine two approaches that have previously been distinct in MPC: 1) global sampling for handling multimodality, and 2) utilizing the dynamic programming principle to transform a high-dimensional estimation problem into connected lower-dimensional ones. In vision-based human tracking, Nonparametric Belief Propagation has been used to sample the marginals corresponding each body part instead of directly sampling the high-dimensional distribution of body model parameters [Sudderth et al. 2010]. In control, dynamic programming has been widely utilized in the variants of Differential Dynamic Programming (DDP). DDP is based on a local second order Taylor expansion of the trajectory, but instead of applying Newton optimization to all trajectory parameters, it performs a sequence of Newton steps to recursively solve the controls for each time instant [Tassa et al. 2014]. DDP is also analogous to belief propagation using Gaussian marginal density models, as exemplified by the AICO algorithm [Toussaint 2009]. Todorov [2008] established a general relation between stochastic control and estimation, suggesting the use of belief propagation. Building on this, Kappen [2012] has also established that optimal control is identical to graphical model inference if the cost can be written as a KL divergence.

As our local refinement pass also uses Gaussian models, our method is related to AICO. The main difference is that we do not build our Gaussian models based on local Taylor expansions, which are problematic with discontinuous and multimodal cost and dynamics. We simply use ensemble estimates for the Gaussian means and covariances, reusing the dynamics simulation results of the forward pass. The estimates are obviously not accurate in many situations, which we handle by not automatically deploying the estimated control; instead, it is evaluated as an initial guess in the next frame. The supplemental video at (00:50) shows how AICO produces smooth results, but in contrast to our method, it fails to switch modes in the multimodal environment of our toy problem. We expect the same to apply to other local trajectory optimization methods including DDP based methods such as iLQG and CC-DDP [Tassa et al. 2012; Tassa et al. 2014]. Considering more complex scenarios, the toy problem is analogous to interactive locomotion with obstacles and leg intercollisions. Changes in steering direction modify the cost landscape, similar to the moving target of the toy problem.

In the context of this paper, we consider the recent work of Tassa et al. [Tassa et al. 2012; Tassa et al. 2014] to represent the state of the art in DDP-based MPC, as 1) they add the handling of box constraints for the controls, which is crucial as real muscles and robot actuators cannot produce infinite forces and torques, and 2) they demonstrate their methods in the control of simulated bipeds, whereas most control theory papers deal with less complex systems. Compared to [Tassa et al. 2012; Tassa et al. 2014], our work adds the abovementioned robustness to multimodality, and the demonstration of a wider variety of movements.

Considering sampling-based control, most of the previous methods draw samples of full trajectories before computing their costs (e.g., [Hämäläinen et al. 2014; Kalakrishnan et al. 2011]). In contrast, our forward pass with resampling informs the control of each simulation step with the previously evaluated costs. Hämäläinen et al. [2014] demonstrate bipedal balancing and recovery only on a simple horizontal plane, and they report that they need a pose-keeping initial guess that makes movement unnaturally stiff. Our system handles a more complex environment and needs no such initial guesses.

The Particle Filter Model Predictive Control (PF-MPC) method [Stahl and Hauth 2011; Kantas et al. 2009] also simulates control forward with resampling. In fact, setting $\mathbf{Q} = 0$ in our method makes the forward pass identical to PF-MPC. The interest in PF-MPC is increasing and it has been applied to, e.g., air traffic management [Eele et al. 2013], but it has not been demonstrated with simulated bipeds. The main difference of PF-MPC in contrast to our method is that information is only propagated forward and there is no backward messages or local refinement.

Considering that the forward simulation with resampling produces a tree of trajectories, both our method and PF-MPC are related to motion planning using the rapidly exploring random tree (RRT) [Lavalle 1998] or the more recent fast marching tree (FMT) [Janson et al. 2013]. They have also been used successfully in simple control problems [Xu et al. 2008]. The methods however operate in state space — RRT samples a new state $\mathbf{x}_{rand}$ and then selects a control $\mathbf{u}$ that minimizes the distance from a nearest state to $\mathbf{x}_{rand}$. In biped control with full dynamics, selecting the $\mathbf{u}$ is nontrivial, which is why we first sample $\mathbf{u}$ and then simulate the corresponding $\mathbf{x}$.

4 Simulation Model and Parameters

For our 3D humanoid tests, we use Open Dynamics Engine (ODE) with the character model shown in Figure 9. The simulation geometry consists of 15 bones (capsules) connected with 3-DOF ball joints except for knees, elbows, and ankles with 1-DOF hinge joints. The model is 1.75m high with a total mass of 70kg. The joints have Euler angle limits manually tuned to match those of an athletic human. Self collisions are enabled between legs but disabled otherwise for best performance. For control, the joints also have ODE angular motors, which take as inputs the desired angular velocities around joint axes and a maximum force that the motor can use to reach the velocity. Inside ODE, the maximum force is

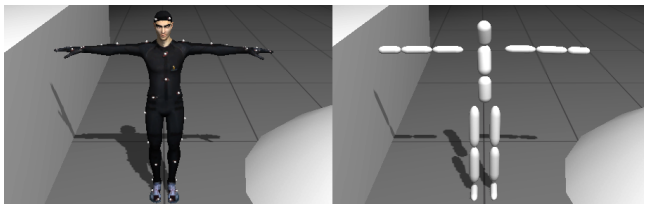

Figure 9: *Our character and the corresponding physics simulation geometry consisting of 15 bones (capsules).*

used as a constraint in the LCP solver. We run the simulation at a fairly large time step of 1/30s. The large time step requires the use of ODE's slower but more stable $\mathcal{O}(N^3)$ direct solver instead of the iterative solver. Selecting the constraint solver and the timestep is a balancing act - the large time step causes some contact instability, but in the end yields a bit better performance than the iterative solver that requires a time step $< 1/100s$ for stability in our case.

We have integrated our C-PBP method with the Unity 3D game engine that was used to produce the visuals in the figures and on the supplemental video. The character mesh is attached to the physics bones using standard skinning.

4.1 Control Parameterization

We use the parameterization $\mathbf{u}_k = [\boldsymbol{\omega}_k, \mathbf{L}_k]$, where $\boldsymbol{\omega}_k$ is a vector of ODE motor target angular velocities for all bones, with a total of 30 elements. $\mathbf{L}_k$ denotes the motor force limits. Here we simplify by optimizing the limits only for 5 bone categories instead of all motors. The categories are left arm, right arm, left leg, right leg, and torso. In total there are 35 optimized control parameters for each forward prediction step k.

We experimented with both joint velocity and torque parameterizations, but in the end found controlling velocities to produce more stable results. For example, balancing can be implemented in many cases by simply setting $\boldsymbol{\omega}_k = 0$. This effectively offloads the low-level torque computations from C-PBP to the ODE solver. The drawback of this approach is that the joints never relax as ODE constantly updates the motor torques needed to reach the $\boldsymbol{\omega}_k$. To allow the joints to relax, we also sample and minimize the $\mathbf{L}_k$ using the zero-mean Gaussian prior in Equation (22).

To prevent occasional high-velocity, high-force glitches due to outlier samples, we also include maximum and minimum value bounds for the $\mathbf{u}_k$. When drawing from the Gaussian mixture or Gaussian resulting from Equation (22), one first selects a Gaussian and then draws the sample from it. After the selection, we use inverse CDF sampling to truncate the Gaussian between the bounds.

4.2 Pose Prior

In all our demos, the character attempts to stay close to some target pose. To prevent the drawing of samples that get discarded due to high deviation from the target, we do not use a quadratic pose deviation cost, but instead add the corresponding Gaussian prior using the $\boldsymbol{\mu}_{k,n}^{(i)}, \mathbf{C}_e$ in Equation (22). Omitting the pose prior produces unnaturally large movement with the joints often being driven to the limits.

Since the motors are parameterized in terms of target angular velocities $\boldsymbol{\omega}$, we set the prior mean as $\boldsymbol{\mu}_k^{(i)} = (\boldsymbol{\zeta}_{\text{target}} - \boldsymbol{\zeta}_k^{(i)})/\delta t$, where $\boldsymbol{\zeta}_k^{(i)}$ denotes the vector of current motor angles (i.e., current pose) and δt is simulation time step. To minimize the number of tuning parameters, we specify the pose prior variance $\text{Cov}(\boldsymbol{\zeta})$ using a single scalar σ_p and the $\mathbf{C_u}$ already used in Equation (22), $\text{Cov}(\boldsymbol{\zeta}) = \sigma_p \mathbf{C_u}$. Assuming transition $\boldsymbol{\zeta}_k = \boldsymbol{\zeta}_{k-1} + \boldsymbol{\omega}\delta t$ we have $\mathbf{C}_e = \sigma_p \mathbf{C_u}/\delta t^2$. This can be verified by considering that $\boldsymbol{\zeta}_{k-1}$ is known at time k, and:

$$\begin{aligned} \text{Cov}(\boldsymbol{\omega}) &= \text{Cov}\left(\tfrac{\boldsymbol{\zeta}_k - \boldsymbol{\zeta}_{k-1}}{\delta t}\right) \\ &= \tfrac{1}{(\delta t)^2}\left(\underbrace{\text{Cov}\left(\boldsymbol{\zeta}_k\right)}_{\sigma_p \mathbf{C_u}} - \underbrace{\text{Cov}\left(\boldsymbol{\zeta}_k, \boldsymbol{\zeta}_{k-1}\right)}_{=0} + \underbrace{\text{Cov}\left(\boldsymbol{\zeta}_{k-1}\right)}_{=0}\right). \quad (30) \end{aligned}$$

Note that the above only applies to the velocity parameters, and the components of $\mathbf{C}_e$ corresponding to the motor force limits $\mathbf{L}$ are set to a large value so that they have no effect.

4.3 Optimizer Parameters

The following lists our settings for the optimizer parameters.

- Number of simulated trajectories (samples) per frame (N), i.e., the sampling budget. A high N increases robustness and decreases movement noise (tremor). We use $N = 32$.
- Planning horizon (K). We use 1.2 seconds for all the behaviors in the supplemental video. Shortening the planning horizon would increase frame rate, but result in less robust, more shortsighted behavior.
- Mutation range $\sigma_m = 0.25$.
- Resampling threshold $T_{eff} = 0.5$.
- The number of "no prior" samples $N_n = 0.25N$.
- Transition noise ($\mathbf{Q}$). We found this difficult to adjust, and finally ended up using $\mathbf{Q} = 0$, although non-zero $\mathbf{Q}$ does improve convergence in the simple example of Section 2. Our problems are probably due to the high dimensionality of the state space combined with the low sample count. We hypothesize that nonzero $\mathbf{Q}$ will be more useful, e.g., in offline optimization with a high sampling budget, which we will explore as future work.

The state $\mathbf{x}_{k,n} \in \mathbb{R}^{36}$ seen by our controller includes a subset of the full ODE simulation state, comprising the 3D position and velocity of the character's pelvis, head, hands and feet. This was found to produce good results with the local refinement backwards pass. The local refinement sometimes produces erroneous results due to the inaccuracies of the Gaussian models but increases movement stability visibly in, e.g., balancing and reaching tasks.

4.4 Movement Parameters

The following gives a list of the movement tuning parameters that a control designer should consider in addition to the basic optimizer parameters above.

- The relative variances of each control variable (diagonal matrix $\mathbf{C}_u$ in Equation (22)). This is the most difficult parameter to tune. Too large variances result in, e.g., hands flailing about all the time or the spine contorting unnecessarily. Too small variances result in sedated movement that is not able to reach goals. We set $\mathbf{C}_u$ to favor large movement of hips and shoulders, and small movements of spine, wrists and ankles.
- The prior scales $\sigma_0, \sigma_1, \sigma_2, \sigma_p$ in Equation (22), denoting joint angular velocity, torque, angular jerk, and pose.

- Sampling bounds. We use a range $[-1.3\pi, 1.3\pi]$ for the angular velocities and $[0, 300]$ for the force limits. Note that these only have effect if σ_0 is large.

4.5 Combining Movement Goals

According to our experiments, our control method is powerful and can make a large variety of movements emerge without tuning parameters for every separate move. However, tuning leads to more natural movement especially regarding the $\sigma_0, \sigma_1, \sigma_2, \sigma_p$ parameters. We have found it useful to model the dynamically changing overall arousal and effort of human movement. For example, one may stand relaxed and calm, but then become tense and aroused if someone pushes one off balance. Sport science has also established that there is an optimal arousal level for each type of movement [Schmidt and Wrisberg 2008]. High effort, low accuracy skills such as weight lifting benefit from high arousal, whereas low effort and high precision skills are easier with low arousal.

We have found that if one does not make the pose, velocity, acceleration priors weaker in high effort movements, the results look too relaxed or even sedated. On the other hand, using the "high effort" settings for basic balancing increases movement noise and makes the character appear tense and twitchy. Some degree of noise is desirable for naturalness as real human movement is not noise-free [Schmidt and Wrisberg 2008].

Based on the considerations above, we have adopted a model where different movement goals such as balance and reach are implemented as software components that interact using the following mechanisms:

- Each goal computes a state cost. The costs are combined additively. The various cost components are explained in detail in the next section.
- Each goal defines a desired effort (arousal) level in the range $[0, 1]$. For example, the "balance in pose" goal prefers high effort when far from balance. At each forward prediction step k, the effort levels are combined using a max function, i.e., the goal desiring the highest effort dominates.
- We specify the tuning parameters $\sigma_0, \sigma_1, \sigma_2, \sigma_p$ separately for high and low effort. The values passed to the optimizer are interpolated linearly between the two based on the overall desired effort. The desired effort has no effect on the state cost.

5 Results

In the following, we refer to supplemental video locations as (mm:ss). All the material in the supplemental video was captured live using Fraps screen capture software with $N = 32$ on a 6 core desktop PC. The target frame rate of 30fps can be achieved by setting $N \approx 24$, but the subjective quality of the results is better at $N = 32$, which decreases movement noise and results in 20-30fps, which still allows fully interactive manipulation of the character using the mouse.

We demonstrate our method for a multitude of behaviors described below together with the cost function design and the parameters. Our results advance the state-of-the-art in terms of variety of movements, especially considering that the optimization runs real time or near-real-time (depending on parameter settings) with no animation data, no pre-computed control policies, and no complex design that becomes brittle in unexpected environments. We also demonstrate novel emergent use of the environment: When balancing on a ball, the character places hands on the walls to stabilize, and tries to climb back on top of the ball after falling (02:48). When trying to keep a ball stable and high, the character automatically moves the ball from head to hand if disturbed (03:25). In the steerable locomotion demo, the character creatively climbs over obstacles.

5.1 Balancing and Recovery

In our balancing demo (00:58), the user may drag the character using left mouse button, or apply a large impulse using the right mouse button. Our character maintains balance when not disturbed and recovers from both the large and small disturbances, taking emergent footsteps and flipping around in air to avoid damage when landing. The behavior emerges from the state cost, which is composed as the sum of several terms:

$$\begin{aligned}\mathbf{s}_k(\mathbf{x}_k) \quad = \quad & \frac{s_{vel}^2}{\sigma_{vel}^2} + \frac{s_{com}^2}{\sigma_{com}^2} + \frac{s_y^2}{\sigma_y^2} + \frac{s_{feet}^2}{\sigma_{feet}^2} + \frac{s_\omega^2}{\sigma_\omega^2} \\ & + \frac{s_{up}^2}{\sigma_{up}^2} + \frac{s_{fwd}^2}{\sigma_{fwd}^2} + s_{damage}. \end{aligned} \tag{31}$$

For brevity, we have dropped the dependency of the terms on $\mathbf{x}_k$. The σ are tuning parameters, which can be interpreted as tolerances. The balancing cost has considerably many terms, but on the other hand, all the other behaviors reuse it and add or disable just a few terms.

s_{vel} is the speed of the center of mass (COM). s_{com} is the horizontal distance of COM from a line connecting the character's feet. s_y penalizes the pelvis bone getting lower than in the target pose relative to the mean of feet, i.e., $s_y = \min(0\text{m}, y_{pelvis} - y_{meanfeet} - d_y)$. $d_y = 1.04$m, which corresponds to the vertical difference between the pelvis and the mean of the feet in the target pose. s_{feet} penalizes feet being too far apart, $s_{feet} = \max(0\text{m}, d_{\text{left\_foot,right\_foot}} - 0.8\text{m})$, where d denotes distance between body parts. s_ω is the angular speed of the pelvis. s_{up} is the difference between the pelvis up vector and the global up vector. s_{fwd} is only used if the user is dragging the character using the mouse, in which case it is the horizontal difference between the head facing direction and the dragging force vector direction. This is motivated by how a real humans tend to look at someone pulling their hand.

$s_{damage} = 10000$ if the character's head touches the environment and zero otherwise. We first used a damage cost proportional to collision velocity, but found out that it is better to categorically forbid head contact, as the character otherwise tends to, e.g., lean on a wall for support using its forehead.

The supplemental video was captured with $\sigma_{com} = 0.025\text{m}, \sigma_y = 0.025\text{m}, \sigma_{vel} = 0.25\text{m/s}, \sigma_{feet} = 0.05\text{m}, \sigma_\omega = 2, \sigma_{up} = 0.1\text{m}, \sigma_{fwd} = 0.1\text{m}$. The pose prior (Section 4.2) favors a basic standing pose.

5.2 Reaching

In our reaching demo (01:26), the user may press a button to activate a glowing sphere that the character tries to reach using left hand. The character is able to take steps and shuffle feet to reach the target. When the target is behind the character, he reorients himself by changing foot positions instead of twisting the torso. When the target is far beyond the reach with an obstacle in between, the character tries to step on the obstacle and jump toward the target. All these behaviors emerge from the balance cost of Equation (31) with two minor modifications: 1) adding the cost term s_r^2/σ_r^2, where s_r

is the distance from hand to target and $\sigma_r = 0.05$m, and 2) increasing the pose prior variance (Section 4.2) for left arm angles. Without the variance adjustment, the reaching goal and the pose prior are in conflict. To maximize naturalness, we also use the s_{fwd} of Equation (31) to make the character look at the target.

5.3 Balancing on Top of a Ball

In the balancing on ball demo (02:13), we remove the s_y and s_ω from the balancing cost and add the terms $s^2_{bcom}/\sigma^2_{bcom}$ and s^2_{by}/σ^2_{by}, where s_{bcom} is the horizontal distance of the character COM from the ball COM, and $s_{by} = \min(0\text{m}, y_{pelvis} - y_{ballTop} - d_y)$. $d_y = 0.9$m, $\sigma_{bcom} = \sigma_{by} = 0.025$m. The additional cost terms are also mapped to the desired effort level. Without this, the character does not attempt to climb back on the ball but rather just stands beside it. The pose prior (Section 4.2) favors a T-pose, which is typical for a human in difficult, unstable balancing.

When commanded to reach the glowing sphere or being pushed on the ball, the character tries to maintain balance by regulating his center of mass or repositioning his feet. This motion causes the ball to roll on the ground, which further increases the challenge of the task. Note that when the ball is nearby a wall, the character autonomously raises his arm to use the wall for support. None of these behaviors are programmed in a state machine or a control policy.

The radius of our ball is 50cm. The character is stable on a heavy ball (250kg), and reaching can be activated even while the ball (02:16). On a light ball (10kg), the character eventually falls off (03:00), but also often eventually climbs back on (03:10).

5.4 Juggling a Ball

In the ball juggling demo (03:18), the character attempts to keep the ball balanced on its head or at least on some body part. This is implemented by augmenting the balance cost with $(\|\mathbf{h}_{\text{head}} - \mathbf{h}_{\text{ball}}\|^2 + \max(0\text{m}, y_{\text{head}} - (y_{\text{ball}} - r))^2)/\sigma^2_{hpos}$, where r is ball radius and $\mathbf{h}$ denotes the horizontal position. The max term causes no cost if head is lower than the ball. If the ball is in contact with the character, the head position is replaced by the contact point for maximum ball handling precision. We use $\sigma_{\text{hpos}} = 0.025$m. When the character is about to lose control over the ball due to external pushes, he quickly switches the strategy from juggling with his head to with one of his arms. When the ball finally falls on the ground, the character kicks the ball up, lets it bounce off the wall, and quickly ducks underneath the ball so that it lands on his head. This complex sequence of actions typically requires sophisticated offline planning, but our algorithm is able to solve it in real-time.

5.5 Steerable Locomotion

In the locomotion demo (03:45), the character can be steered with the analog stick of a gamepad. By simply replacing the s_{vel} term in Equation (31) with a similar term that minimizes the difference from the target velocity, here $1.25m/s$ in the direction of the analog stick, we are able to repurpose a static balance controller to a locomotion controller. We admit that the locomotion is not beautiful, but this was done without any prior knowledge of posture or timing of human gait.

The character handles sudden turns and small obstacles for foot placement but stops in front of large obstacles. However, the user can press a gamepad button to activate a "turbo" mode (04:19) that allows the character to discover ways to climb over obstacles. This is simply done by disabling the s_{up} and s_{com} terms of the balancing cost, setting desired effort level to maximum, and adjusting the target velocity slightly upwards. Disabling the terms is needed as the climbing may require leaning on top of the obstacle with torso not upright and COM not directly above feet.

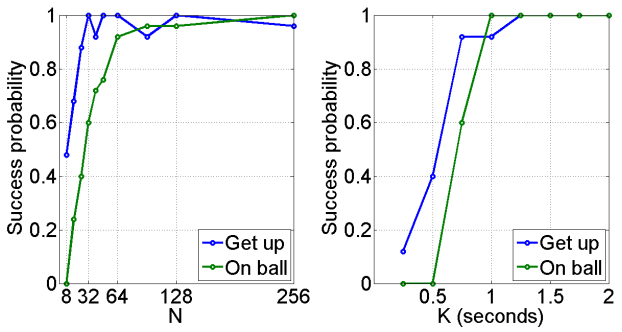

Figure 10: *Success probabilities in getting up and balancing on a moving ball, with K=1.2 and varying N (left), and N=256 and varying K (right).*

5.6 Parameter Exploration

The sampling budget N and planning horizon K are the most critical parameters for both robustness and computing cost. To explore their effect, we have run a series of simulations with different N and K and measured the average probability of success. We used 25 repetitions of two trials with no user interaction to control for other than the manipulated parameters: A) the balancing objective of Section 5.1 with the character initialized in a supine T-pose so that it has to both get up and balance, and B) the balance-on-ball objective of Section 5.3, with the character dropped on a ball that is initially rolling to require emergent locomotion on the ball. Success was defined as the state cost being lower than a threshold at 5 seconds from the start of the test. The thresholds were adjusted so that the character must be stable but some deviation from the target pose is allowed. The results are illustrated in our parameter exploration video (part of the auxiliary material) as well as in Figure 10. The figure indicates that increasing N and/or K improves robustness and both parameters must be large enough for any chance of success. With our default $N = 32$, $K = 1.2$, doubling N improves the success rate from 0.6 to 0.92 in task B. $K > 1$ appears to be required for task B, but on the other hand, our subjective experience is that $K = 2$ starts increasing movement noise, apparently due to the increased number of recursions decreasing the accuracy of Algorithm 1. The video visualizes the results of Algorithm 1 using a yellow trajectory for the pelvis, which can be observed to deviate from the current best trajectory (green) more often with high K. The sampled trajectories are shown in grey.

5.7 Limitations

The main limitation of our system is that it is only barely real-time, and one would like to both increase the sample budget N and planning horizon K to improve the movement quality. Presently, movement is slightly stiff and noisy. Stiffness and noise can be reduced by decreasing the prior variances for the motor force limits and angular velocities. This however also decreases the success rate of recovering from large disturbances if one does not compensate by increasing N and K. Fortunately, our method parallelizes trivially, and can thus benefit from future increases in computing power.

Our character model is also too simple, e.g., for more complex object manipulation tasks. As the physics geometry does not model hand details, the character often makes heavy contact with the en-

vironment using the end of the hand capsule, i.e., fingertips. This looks unnatural, which we have fixed by manually overriding the sampled wrist controls if the hand is approaching a solid object at high velocity. In addition to the simple skeletal model, we are also using simple motors instead of more detailed muscle models, which makes it unlikely that fully natural movement would emerge even with unlimited sampling budget.

Another important limitation is that we are not modelling the limits of human perception. The forward simulation gives the character uncannily perfect situational awareness. For example, without the explicit guide to look at the mouse cursor (the s_{fwd} in Equation (31)), the character does not bother turning if dragged backwards, as it does not need to see the ground to place a hand for support.

6 Conclusion

We have presented a novel sampling-based method for online control of simulated characters. The key components of the sampling are a forward pass with resampling similar to previous work on Sequential Monte Carlo and particle filters, augmented with backwards message passing and recursive local refinement of the control trajectory. Considerable care was also put to including as many cost function components as possible in the proposal distribution, which minimizes redundant sampling and simulation of high-cost trajectories. In our case, the proposal includes minimization of velocity, acceleration, and jerk as well as the difference from a target pose.

Our results demonstrate a large variety of interactive behaviors that emerge from adding or disabling a few terms to a basic balancing cost function. Our behaviors are robust and adaptive - the character recovers automatically from large and small disturbances, and uses the environment when applicable. This provides new opportunities for both animation research and the development of interactive experiences such as games.

One clear future development direction is online learning and gradual improvement of the optimization results, e.g., to achieve higher quality locomotion without sacrificing the adaptiveness and interactivity. We hypothesize that we do not need to learn whole trajectories but only lower-dimensional marginal density priors that can be plugged in similar to our Gaussian mixture based on the previous frame (Equation (25)). We also aim to test our method with more detailed hand geometry as well as in offline motion optimization.

Considering the limitations outlined in Section 5.7, more detailed models of human perception need to be developed for improved realism. The information produced by the forward simulation has to be filtered to contain only such predictions that real humans could formulate based on their senses and mental models. This is a fascinating topic, and we hypothesize that it might be reasonable to not produce the information in the first place, e.g., terminate a simulation trajectory early and save computing resources if there is an event that will make the rest of the trajectory be affected by information not available through realistically simulated senses.

Acknowledgments

We thank all the reviewers for their valuable comments. This work was in part funded by NIH R01 NS069655 as well as the Skene - Games Refueled program of the Finnish Funding Agency for Innovation.

References

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

BORNO, M. A., FIUME, E., HERTZMANN, A., AND DE LASA, M. 2014. Feedback control for rotational movements in feature space. *Comput. Graph. Forum 33*, 2, 225–233.

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

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Comput. Graphics Forum 27*, 2, 371 – 380.

EELE, A., MACIEJOWSKI, J., CHAU, T., AND LUK, W. 2013. Parallelisation of Sequential Monte Carlo for real-time control in air traffic management. In *Proc. CDC 2013*, 4859–4864.

GEIJTENBEEK, T., AND PRONOST, N. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graphics Forum 31*, 8, 2492 – 2515.

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.

GUO, S., SOUTHERN, R., CHANG, J., GREER, D., AND ZHANG, J. 2014. Adaptive motion synthesis for virtual characters: a survey. *The Visual Computer*, 1–16.

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

HÄMÄLÄINEN, P., ERIKSSON, S., TANSKANEN, E., KYRKI, V., AND LEHTINEN, J. 2014. Online motion synthesis using sequential monte carlo. *ACM Trans. Graph. 33*, 4 (July), 51:1–51:12.

IHLER, A. T., AND MCALLESTER, D. A. 2009. Particle belief propagation. In *Proc. International Conference on Artificial Intelligence and Statistics*, 256–263.

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

JANSON, L., CLARK, A., AND PAVONE, M. 2013. Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions. *arXiv preprint arXiv:1306.3532*.

KALAKRISHNAN, M., CHITTA, S., THEODOROU, E., PASTOR, P., AND SCHAAL, S. 2011. STOMP: Stochastic trajectory optimization for motion planning. In *Proc. ICRA 2011*, IEEE, 4569–4574.

KANTAS, N., MACIEJOWSKI, J., AND LECCHINI-VISINTINI, A. 2009. Sequential monte carlo for model predictive control. In *Nonlinear Model Predictive Control*. Springer, 263–273.

KAPPEN, H. J., GÓMEZ, V., AND OPPER, M. 2012. Optimal control as a graphical model inference problem. *Machine learning 87*, 2, 159–182.

LAVALLE, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. rep., Iowa State Univ.

LEE, Y., KIM, S., AND LEE, J. 2010. Data-driven Biped Control. *ACM Trans. Graph. 29*, 4, 129:1–129:8.

LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012. Terrain Runner: Control, Parameterization, Composition, and Planning for Highly Dynamic Motions. *ACM Trans. Graph. 31*, 6, 154:1–154:10.

MORDATCH, I., DE LASA, M., AND HERTZMANN, A. 2010. Robust Physics-based Locomotion Using Low-dimensional Planning. *ACM Trans. Graph. 29*, 4, 71:1–71:8.

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.

PEJSA, T., AND PANDZIC, I. 2010. State of the Art in Example-Based Motion Synthesis for Virtual Characters in Interactive Applications. *Comput. Graphics Forum 29*, 1, 202–226.

SCHMIDT, R. A., AND WRISBERG, C. A. 2008. *Motor Learning and Performance*, 4rd ed. Human Kinetics.

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

SUDDERTH, E. B., IHLER, A. T., ISARD, M., FREEMAN, W. T., AND WILLSKY, A. S. 2010. Nonparametric belief propagation. *Commun. ACM 53*, 10, 95–103.

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

TASSA, Y., EREZ, T., AND TODOROV, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proc. IROS*, 4906–4913.

TASSA, Y., MANSARD, N., AND TODOROV, E. 2014. Control-limited differential dynamic programming. In *Proc. ICRA 2014*, IEEE, 1168–1175.

TODOROV, E. 2008. General duality between optimal control and estimation. In *Proc. CDC 2008*, 4286–4292.

TOUSSAINT, M. 2009. Robot trajectory optimization using approximate inference. In *Proc. ICML 2009*.

WAMPLER, K., POPOVIĆ, Z., AND POPOVIĆ, J. 2014. Generalizing locomotion style to new animals with inverse optimal regression. *ACM Trans. Graph. 33*, 4 (July), 49:1–49:11.

WITKIN, A., AND KASS, M. 1988. Spacetime Constraints. *SIGGRAPH Comput. Graph. 22*, 4, 159–168.

WU, J.-C., AND POPOVIĆ, Z. 2010. Terrain-adaptive bipedal locomotion control. *ACM Trans. Graph. 29*, 4, 72:1–72:10.

XU, J., DUINDAM, V., ALTEROVITZ, R., AND GOLDBERG, K. 2008. Motion planning for steerable needles in 3d environments with obstacles using rapidly-exploring random trees and backchaining. In *Proc. CASE 2008*, IEEE, 41–46.

YE, Y., AND LIU, C. K. 2010. Optimal Feedback Control for Character Animation Using an Abstract Model. *ACM Trans. Graph. 29*, 4, 74:1–74:9.

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

Algorithm 2 Control Particle Belief Propagation (C-PBP) for frame n.

Input: Initial state $\mathbf{x}_{0,n}$, previous trajectories $\mathbf{u}^{(i)}_{1\dots K,n-1}$, previous smoothed trajectory $\widehat{\mathbf{u}}_{1\dots K,n-1}$, previous best trajectory $\mathbf{u}^{(b)}_{1\dots K,n-1}$, state and control cost functions $s^{(i)}_{k,n}(\mathbf{x}^{(i)}_{k,n})$, $c^{(i)}_{k,n}(\mathbf{u}^{(i)}_{k,n})$, transition function $f(\mathbf{x},\mathbf{u})$, number of forward planning steps K, number of samples N, number of "no prior" samples N_n, resampling threshold T_{eff}, transition covariance $\mathbf{Q}$, prior parameters $\mathbf{C}u, \sigma_0, \sigma_1, \sigma_2$
Output: Optimal control trajectory $\mathbf{u}^{(b)}_{1\dots K,n}$, smoothed trajectory $\widehat{\mathbf{u}}_{1\dots K,n}$, sampled trajectories $\mathbf{u}^{(i)}_{1\dots K,n}$

//The forward pass
for $k = 1 \dots K$ **do**
  //resampling
  Compute N_{eff} according to Equation (18)
  if $N_{\text{eff}} < T_{\text{eff}} N$ **then**
    Resample the continued trajectories
  end if
  //sampling
  for $i = 1 \dots N$ **do**
    //Draw each sample i
    if $i == 1$ **then**
      //Initial guess: Previous best trajectory
      Set $\mathbf{u}^{(i)}_{k,n} = \mathbf{u}^{(b)}_{k+1,n-1}$
    else if $i == 2$ **then**
      //Initial guess: Previous smoothed trajectory
      Set $\mathbf{u}^{(i)}_{k,n} = \widehat{\mathbf{u}}_{k+1,n-1}$
    else if $i < 2 + N_n$ **then**
      Draw $\mathbf{u}^{(i)}_{k,n}$ according to Equation (22),
      but without $\mathcal{P}(\mathbf{u}^{(i)}_{k,n} | \mathbf{z}^{(:)}_{:,n-1})$.
    else
      Draw $\mathbf{u}^{(i)}_{k,n}$ according to Equation (22).
    end if
    //Simulate the system forward
    Set $\mathbf{x}^{(i)}_{k,n} = f(\mathbf{x}^{(h(i,k,-1))}_{k-1,n}, \mathbf{u}^{(i)}_{k,n})$
  end for
  Compute state potentials according to Equation (21).
  Compute forward beliefs according to equations (16), (17), omitting $\widehat{B}_{\text{fwd}}(\mathbf{z}^{(i)}_{k-1})$ if we resampled at this k.
end for
//Backward message passing and smoothing
Compute backwards messages and beliefs using equations (15), (13)
Obtain smoothed control $\widehat{\mathbf{u}}_{1\dots K,n}$ using Algorithm 1
//Finish
Deploy the optimal control $\mathbf{u}^{(b)}_{1,n}$, $b = \arg\max_b(\mathcal{P}(\mathbf{z}^{(b)}))$
return $\mathbf{u}^{(b)}_{1\dots K,n}$, $\widehat{\mathbf{u}}_{1\dots K,n}$ and $\mathbf{u}^{(i)}_{1\dots K,n}$