EG UK Computer Graphics & Visual Computing (2018)
G. Tam and F. Vidal (Editors)

Keys-to-Sim

Transferring Hand-Crafted Key-framed Animations to Simulated Figures using Wide Band Stochastic Trajectory Optimization

Dominik Borer[1,2] Martin Guay[1] Robert W. Sumner[1,2]

[1]Disney Research [2]ETH Zürich

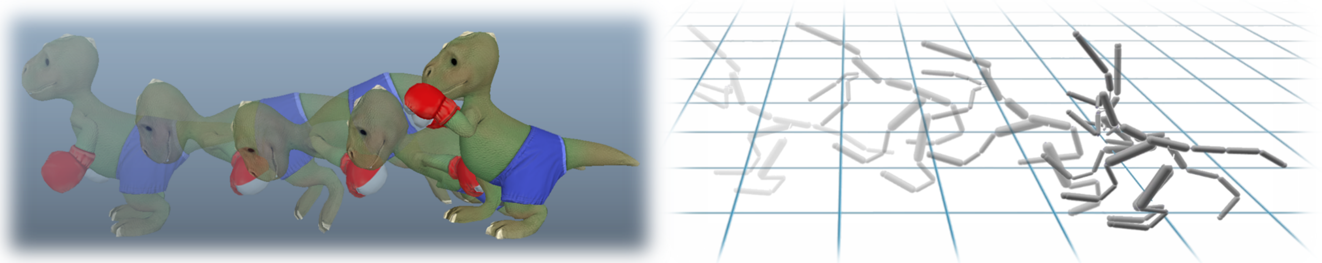

Figure 1: *With our method we can transfer hand-crafted key-framed animations to a simulated character as demonstrated with the cartoon dinosaur, performing a physically infeasible jumping motion. Transferring such movements to a simulated character requires a large deviation from the original motion—which is not supported by previous methods that support only a narrow search band for optimization.*

Abstract

The vision of fully simulating characters and their environments has the potential to offer rich interactions between characters and objects in the virtual world. However, this introduces a challenging problem similar to controlling robotic figures: computing the necessary torques to perform a given task. In this paper, we address the problem of transferring hand-crafted kinematic motions to a fully simulated figure, by computing open-loop controls necessary to reproduce the target motion. One key ingredient to successful control is the mechanical feasibility of the target motion. While several methods have been successful at replicating human captured motion, there has not yet been a method capable of handling the case of artist-authored key-framed movements that can violate the laws of physics or go beyond the mechanical limits of the character. Due to the curse of dimensionality, sampling-based optimization methods typically restrict the search to a narrow band which limits exploration of feasible motions—resulting in a failure to reproduce the desired motion when a large deviation is required. In this paper, we solve this problem by combining a window-based breakdown of the controls on the temporal dimension, together with a global wide search strategy that keeps locally sub-optimal samples throughout the optimization.

CCS Concepts

•***Computing methodologies*** $\rightarrow$ ***Physical simulation;*** *Optimization algorithms;*

1. Introduction

While feedback controllers have been created in the past for various motions such as walking, running, swimming, aerial stunts and bicycle riding [HWBO95, YLvdP07, TGTL11, HYL12, TGLT14], they were designed on a case-by-case basis often using motion-specific modeling, and cannot be used to produce other movements. One way to generalize control to various movements is to learn from examples [LYvdP*10, LYG15, PBYvdP17]. While much progress has been made over the past years on transferring human captured motion to the simulation world, the methods fall short when it comes to hand-crafted key-framed animations. The main reason is the large search band required to explore deviations from the desired motion, while at the same time avoiding the curse of dimensionality.

For example, consider the case of a walk cycle. The character could immediately start walking from the first frame, having the back leg moving forward without any center-of-mass momentum—causing the character to fall on its back. A mechanically feasible version of such a movement would require a large deviation from the desired motion. Hence, previous methods such as frame-by-frame inverse-dynamics [AdSP07], and sampling-based optimization [LYvdP*10, ABdLH13], cannot be used successfully in the context of hand-crafted key-framed motions, due to their narrow search space.

In this paper, we solve this problem with a wide band sampling-based trajectory optimization method. Given an artist-crafted animation, our algorithm outputs mechanically feasible movements (open-loop controls). To support a large coverage of the parameter space, our algorithm relies on two key innovations. First, we break down the controls into a sequence of overlapping control windows, each typically of 0.5 seconds length. While this helps to converge locally more successfully, it does not support finding global solutions in the long run. Hence, our second key innovation is to keep multiple samples and thus multiple sequences of control windows alive, to finally select the best sequence of samples at the end—by backtracking the entire process. We demonstrate the success of this strategy compared to previous approaches in our accompanying video.

Another key ingredient to the success of trajectory optimization methods is the design of the mechanical figure. For example, the overall mass of the character will often be very light, except for its abdominal area (pelvis). The joint type (dimensionality) and PD (proportional-derivative) stiffness and damping coefficients will often be tweaked for each motion, thereby reducing the search space of the optimization. These aspects can be mind-boggling for casual users, which our system is designed for. In this paper, we describe a modeler that takes as input a kinematic skeleton and yields a control-friendly character with minimal user interactions.

To summarize, our contributions are as follows:

- A wide band sampling-based trajectory optimization algorithm to compute mechanically feasible open-loop controls from hand-crafted key-framed animations (Section 5).
- A mechanical system modeler that abstracts the insights required for creating a control-ready character from a kinematic skeleton (Section 4).

2. Related Work

Physics-based character animation has a long history in computer animation and robotics. We focus on the works most related to trajectory optimization and policy learning, while referring the reader to the recent survey of Geijtenbeek et al. [GP12] for a wider inspection.

Early controllers were designed manually for specific motions using a combination of PD-control and virtual forces: human athletics and running [HWBO95], locomotion [YLvdP07, WFH09, LKL10, CKJ*11], as well as swimming and aerial stunts [TGTL11, HYL12]. For more general control methods, researchers have looked at methods for tracking example motions with a simulated character.

Model-based methods utilise the full equations of motion to optimize for controls—for a single time step [AdSP07, MZS09, MdLH10, dLMH10, YL10, RvdPK14], or a whole trajectory [WK88, MTP12]—using local optimization such as quadratic programming or BFGS. While these methods support tracking example motions to some extent, they are inherently tied to specific mechanical modeling—in generalized coordinates—and transferring the controls to a simulator operating in maximal coordinates remains an issue. Additionally, coping with discontinuous contact dynamics requires intricate modeling for compatibility with local gradient-based optimization.

Motivated by these problems, several works focused on model-free and gradient-free optimization techniques, where the simulator (in maximal coordinates) is treated as a black box [LYvdP*10, ABdLH13, HL14, HET*14, LYG15, HRL15, RH17, NRH17]. In these works, sample controls (torques or PD joint targets) are simulated over a time window, and optimized using various sampling schemes. While some can synthesize motions without example movements [ABdLH13, HL14, HET*14, HRL15, RH17, NRH17], they require specific objective functions for each motion. Methods designed for tracking an example motion [LYvdP*10, LYG15], have focused mainly on feasible realistic human captured motion. In the case of hand-crafted motions, these methods fail to converge due to the limited scope of the optimization space considered. In contrast, our sampling scheme supports a larger deviation from the original example motion, while keeping a single set of objective functions.

The approaches mentioned in the previous paragraph generate open-loop controls, which are time-dependent control values to be used in the same sequence of states they were computed for. In contrast, a state-dependent feedback policy function covers a portion of the character's state space and can adjust to different situations. Surprisingly simple feedback policies such as factored linear functions were shown capable of handling specific tasks such as balancing, and walking [DLvdPY15], and full rank linear functions have successfully been used for parkour movements [LYvdPG12]. Linear functions alone however are not sufficiently powerful to model the complex relations between the character's state and the required controls across multiple tasks. For example, transitioning from idle to walking and then running has been broken down into different linear feedback controllers in [LYvdPG12, LvdPY16].

The recent success with training deep neural networks has brought a re-visitation of reinforcement learning with regard to learning continuous feedback policies. The idea of deep Q-learning has been combined with valid open-loop controls in the case of humanoid skateboard balancing and riding [LH17]. A continuous actor-critic model has been successfully trained to map high-dimensional terrain data to controls of a 2D quadruped [PBvdP15, PBvdP16], and more recently a hierarchical RL scheme has been successfully trained for 3D biped locomotion [PBYvdP17] using example human motion. The recent case of 3D biped locomotion [PBYvdP17] has been successfully trained without having to first initialize the policy with valid controller samples. However, the method requires a realistic and *feasible* human reference motion. Given the general unstable convergence rate of RL algorithms (e.g. trust region policy optimization and continuous actor-critic), we conject that without physically feasible reference motions, convergence could only be worse and most likely fail—hence the need for feasible movements and the relevance for our method to generate such movements from hand-crafted animations.

Control-ready Character Modeler One of the key ingredients for control algorithms is a well designed mechanical system for the character. The characters are often very light to reduce control magnitudes. The mass is often concentrated at the pelvis to improve stability of the overall system's dynamics. In many cases, the PD stiffness and damping coefficients are specific to certain motions to reduce the optimization search space. To our knowledge, tools that include these insights directly to create a mechanical system for the character have never been presented. As a consequence, operating a simulated character requires intricate engineering knowledge. In [CBvdP10], the user can adjust the proportions of a biped, but new characters such as a quadruped must be engineered. While in [LWH*12] it is mentioned that the articulated rigid body system is automatically generated from the kinematic skeleton of the character, it lacks details—for example how joint configurations as in Figure 3 are handled—which we provide in this paper.

3. Overview

Our trajectory optimization algorithm optimizes the controls of a simulated character, while digital artists typically design kinematic motions using a kinematic skeleton (KS). Unfortunately, the kinematic skeleton cannot be used directly for simulation and must be converted into an articulated rigid body system (ARBS).

Because each aspect of the ARBS (body shapes, masses, joint types, and PD stiffness and damping coefficients) can each influence the stability and feasibility of the control, we provide a modeler (Section 4) to facilitate creating it from a kinematic skeleton. The user simply drag-and-drops the kinematic skeleton and with a few edits recovers a fully modeled ARBS.

The next step is to provide the trajectory optimization with a target motion. Because joint dimensionalities can be different between the ARBS and KS representations (e.g. hinge joints), we first transfer the set of key-frames in the KS to a set of key-frames in the ARBS representation, using full body inverse kinematics.

We then convert these target motions into mechanically feasible movements using our wide band sampling-based trajectory optimization algorithm (Section 5). This optimization results in open-loop controls (PD joint angles) for a sequence of overlapping control windows (of 0.5 seconds each). The final motion is created by sampling the controls over the first half of each window sequentially over time.

4. Articulated Rigid Body Modeler

The user provides a character skinned with a *Kinematic Skeleton* (KS), which lacks mechanical information such as rigid body shapes and masses, joint types and PD stiffness and damping coefficients. To use the character within a simulation, we first need to set up a corresponding rigid body and joint structure, with appropriate mechanical properties, which is referred to as *Articulated Rigid Body System (ARBS)* (Figure 2). We observed that it was easier to prune an existing ARBS, than model one from scratch. Hence when the user drag-and-drops a KS into our modeler, our system first creates an initial ARBS automatically. To create this ARBS special care has to be taken to create the proper rigid bodies (Section 4.1) and joints (Section 4.2) in order to mimic the motion space of the KS. Due to missing information in the given KS the created ARBS might have undesired artifacts, such as wrongly oriented rigid bodies or joint axes, which can be tweaked by the user with a few interactions (Section 4.3).

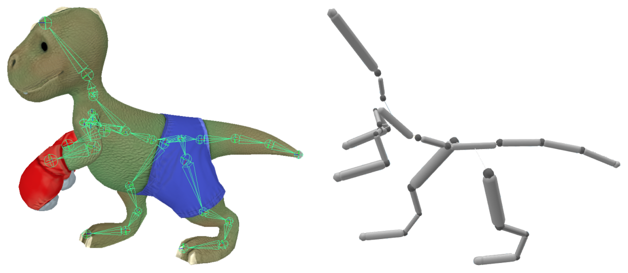
Figure 2: *Using our modeler, the user-designed cartoon dinosaur character with its kinematic skeleton (KS) is easily converted to the corresponding articulated rigid body system (ARBS).*

4.1. Creating Rigid Bodies

Directly mapping the kinematic joints to the simulated joints, with rigid bodies in-between, leads to a structure with more degrees of freedom (DoF) than the KS allows (Figure 3(b)). By observing the function of the kinematic joints, we see they are in fact rigid bodies with full 3D rotation and several children (whereas ARBS joints have only one parent and child). Thus to mimic the motion space of the KS with the ARBS, we associate a rigid body to a KS joint and create simulation joints between the rigid bodies (Figure 3(c)).

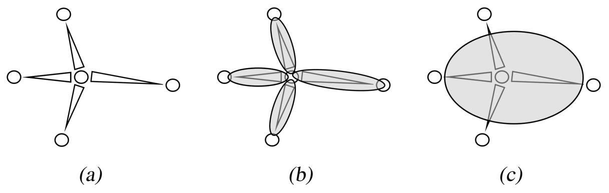
Figure 3: *The kinematic skeleton (a) is represented through circles for the joints and connections pointing from the parent to the child joint. Rigid bodies are represented with gray ellipses. Creating a rigid body per connection (b) results in too many degrees of freedom whereas associating a rigid body with a skeleton joint (c) leads to the proper structure.*

Preserving the connections to all child skeleton joints, leads to the following three different configurations:

No Child Joint: A KS joint without children marks the endpoint of a hierarchy. These KS joints are usually not animated and not associated with any bone, and thus no rigid body is created.

Exactly one Child Joint: KS joints with exactly one child joint represent simple bones as we see in arms or legs. For those, a rigid body with its endpoints at the two KS joints is created.

More than one Child Joint: KS joints with more than one child joint represent more complex bones as in Figure 3 (e.g. pelvis or upper body). For these, we need to create a rigid body that covers all the associated joints. To get an estimate of the orientation, we perform a principal component analysis on the KS joint positions and orient the rigid body along the principal direction.

Rigid Body Properties Before a rigid body can be used in a simulation, it needs to be set up properly. As colliders we use capsules, with the radius set to 10% of its length. The mass is estimated based on the volume of the capsule—assuming a uniform distribution—while the moment of inertia is approximated from a box enclosing the capsule.

4.2. Creating Joints

The constraints between the rigid bodies are imposed through joints. Because we want to keep the dimensionality as low as possible—to help optimization convergence—we use only hinge joints with 1 DoF. Since the artist is not constrained when creating the character, the KS usually comes with 3 DoF per joint. This introduces an ambiguity: which rotation axis to use for the hinge joints. We make a guess and let the user edit the axis. Our guess is defined as the direction orthogonal to the directions from the joint position to the two associated rigid bodies.

4.3. Editing the ARBS

Although the automatically generated ARBS is estimated from the KS in a reasonable fashion, it requires additional tweaks to yield best results; for example, with the orientation of the hinge joints, or the need for end-effectors for stable ground contacts. Our modeler thus provides the following set of tools to quickly tweak the ARBS:

- Rigid bodies: change position, orientation, size and mass.
- Joints: change rotation axis and limits.
- End-effectors: add to a terminal rigid body and change its position and size.
- Structure: select new root, delete or merge rigid bodies.

With the final ARBS ready, we transfer the target motion keyframes from the KS to the ARBS representation using full body inverse kinematics. Given the desired motion in the same representation, we turn to optimizing for the controls that can mimic the desired motion, but in a simulated environment.

5. Wide Band Stochastic Trajectory Optimization

The simulated character state $\mathbf{Q}$ is controlled through PD-control using *control poses* $\mathbf{Q}_c$ (a set of joint angles). Naive tracking of the target poses ($\mathbf{Q}_c = \mathbf{Q}_t$) through PD-control does not work and the character quickly falls. The goal of the wide band trajectory optimization is to compute control *offsets* $\Delta\theta$ for the control pose $\mathbf{Q}_c' = \mathbf{Q}_t + \Delta\theta$, such that the simulated motion $f_{sim}(\mathbf{Q}_c')$ is similar to the target motion as illustrated in Figure 4.

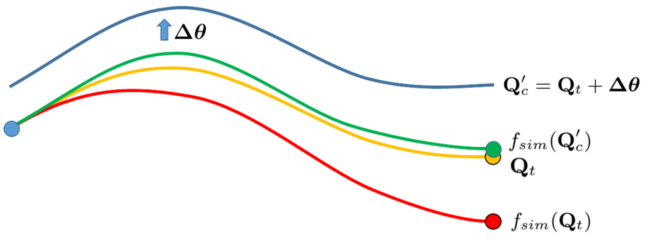

Figure 4: *The goal is to reproduce the target curve $\mathbf{Q}_t$ (orange) through the simulation. Using $\mathbf{Q}_t$ as the control pose results in $f_{sim}(\mathbf{Q}_t)$ (red), which is far away from the goal. Adding offsets $\Delta\theta$ to the target curve and use the control pose $\mathbf{Q}_c' = \mathbf{Q}_t + \Delta\theta$ leads to a much better result $f_{sim}(\mathbf{Q}_c')$ (green). The simulation does usually not exactly match because the character has to maintain balance or build up momentum which may contradict with the target motion.*

During the simulation, the torques τ driving the joint orientation $\mathbf{q}$ and angular velocity ω towards the control orientation $\mathbf{q}_c$ and angular velocity ω_c are computed through PD-control as:

$$\tau = k_p \cdot D_{\mathbf{q}}(\mathbf{q}_c, \mathbf{q}) + k_d \cdot D_{\omega}(\omega_c, \omega),$$

where k_p and k_d are joint stiffness and damping coefficients and $D_{\mathbf{q}}$ and D_{ω} are appropriate distance functions.

The control offsets are broken down into time steps of 0.1s while the simulation runs at a time step of 0.0005s, using linear interpolation in-between to sample the controls. Optimizing the full space and time of the motion suffers from the curse of dimensionality. Hence we break down the controls into different temporal windows that quickly converge locally. To obtain a wide search band in the parameter space, we sample from a Gaussian distribution which is adapted over multiple optimization iterations, and keep multiple samples per control window alive over time. The best solution is then recovered by backtracking the results from the end state. The entire algorithm is summarized as pseudocode in Algorithm 1.

Algorithm 1 Wide Band STO

```
 1: procedure TrajectoryOptimization($n$, $k$, $N_{CMA}$)
 2:     $n$   (number of samples to generate per window)
 3:     $k$   (number of samples to keep per window)
 4:     $N_{CMA}$   (number of CMA-ES iterations for each initial condition)
 5:     $n_s \leftarrow \frac{n}{k}$   (number of samples to generate for each initial condition)
 6:     $\mathbf{S}_{start} \leftarrow \mathbf{s}_{init}$ $k$-times   (character states at begin of window)
 7:     $\Psi_{selected} \leftarrow$ empty   (samples kept alive per window)
 8:     $id_{window} \leftarrow 0$   (current window index)
 9:     while motion not finished do
10:         $\Psi_{generated} \leftarrow$ GenerateSamples($\mathbf{S}_{start}, n_s, N_{CMA}$)
11:         $\Psi_{selected}[id_{window}] \leftarrow$ SelectSamples($\Psi_{generated}, k$)
12:         $\mathbf{S}_{start} \leftarrow$ SimulateHalfWindow($\mathbf{S}_{start}, \Psi_{selected}[id_{window}]$)
13:         $id_{window} = id_{window} + 1$
14:     end while
15:     $\kappa \leftarrow$ BacktrackFinalTrajectory($\Psi_{selected}$)
16: end procedure
```

5.1. Control Windows

While optimizing for the offset poses for the entire motion is not possible due to the curse of dimensionality, sequentially optimizing for small control windows of 0.1s is too short sighted and lacks context. We conject that such an approach is successful only when applied with close-to-feasible motions such as human captured motion, as in [LYvdP*10]. To provide enough far-sightedness and enrich the control windows with context of close-by states, we split the controls into larger windows of 0.5s length that overlap each other half-way (every 0.25s). This way the optimization has a 0.25s takeoff to be compatible with the previous window, and 0.25s exploration that the next window will start from. The optimization is then performed over the whole motion following an advancing schedule, as shown in Figure 5.

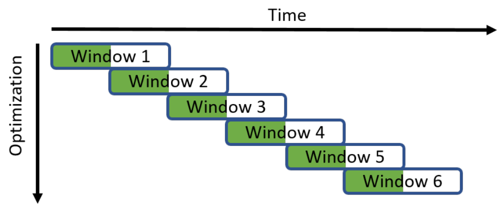

Figure 5: *To provide enough far-sightedness while still keeping the dimensionality feasible, the optimization considers a shorter control window containing only a few offset poses. Using overlapping control windows allows to take goals of the next window into account which ensures good initial conditions for the next window.*

5.2. Sampling Scheme

For each control window, we generate n random samples from a multivariate Gaussian distribution, where each sample is a set of offset poses within the control window. The samples are then evaluated by running the simulation through the entire window and computing a cost function (1) comparing the resulting simulated motion to the target motion. Using these cost estimates, the sampling distribution is adapted towards the better samples, and the process is repeated for several iterations before advancing to the next control window.

Optimizing the window locally to a single optimum and moving to the next window in a greedy fashion, can however lead to the character diverging and falling in the long run, since the optimization is agnostic to the global optimum. We address this problem by keeping multiple samples $k << n$ for each control window alive throughout the global process, and advancing all of them to the next window, to recover the final globally optimal solution, once the last window is reached. In total, we generate n samples for each window, and then select $k << n$ of those to advance to the next window, leading to k initial conditions.

When optimizing the next window, for each initial condition we generate $n_s = \frac{n}{k}$ samples to keep the total number of samples bounded, as illustrated in Figure 6. To support a wide search band in a sample-efficient manner, we employ a distribution adaptation scheme. Specifically, we use *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)* [Han06] to adapt the distribution of each "alive" sample, during the subsequent window optimization. As can be seen in Figure 7, the globally best solution is often not locally optimal, which shows that keeping samples of earlier iterations of CMA-ES is beneficial.

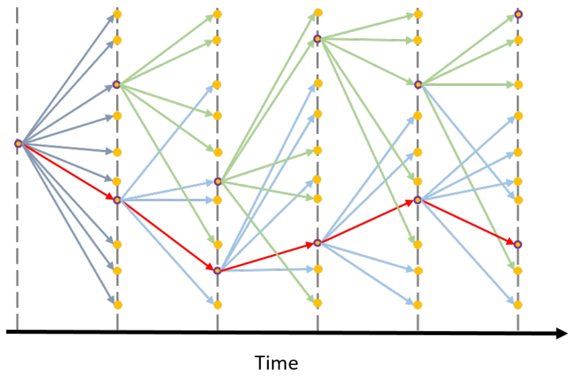

Figure 6: *Example of the sampling process for $n = 10$ and $k = 2$. Samples from different initial conditions are represented in different colors. The final best trajectory is shown in red.*

We used the following parameters during the CMA-ES window optimizations: we set the initial mean and standard deviation to $\mu = 0$ and $\sigma = 0.01$. Then we run CMA-ES for 200 steps each with a population of size 30. For all experiments we generated $n = 400$ samples per window from which we kept $k = 20$ to proceed to the next window. This means for each of the k initial conditions in the next window we generate $n_s = 20$ samples, which means that during the 200 steps of CMA-ES, every 10 steps the current best solution is stored as a sample.

5.3. Evaluating Samples: Cost Functions

To adapt the sampling distribution, we evaluate the samples using cost functions comparing the resulting motion to the target motion. The samples are a sequence of pose offsets (control poses) that we use to move the character forward in the simulation window. While the simulation of the character runs at a higher frequency (0.0005s), we compute the cost function at a low frequency (0.05s).

Because the target motion is not necessarily feasible, we need cost functions that are not strictly measuring the similarity between poses. Instead we compute a loose measure of similarity that emphasizes feature points such as the end-effectors and global root position, while weighting the pose similarity less. Hence the final cost energy to minimize is:

$$E_{window} = E_{root} + E_{pose} + E_{ee}. \tag{1}$$

Root E_{root} measures the global position and orientation:

$$\begin{aligned} E_{root} &= E_h + E_{\mathbf{p}} + E_{\mathbf{q}}, \\ E_h &= w_h \cdot |h_{target} - h_{sim}|^2, \\ E_{\mathbf{p}} &= w_{\mathbf{p}} \cdot \|\mathbf{p}_{target} - \mathbf{p}_{sim}\|_2^2, \\ E_{\mathbf{q}} &= w_{\mathbf{q}} \cdot \|\texttt{axisAngle}(\mathbf{q}_{target}^{-1} \cdot \mathbf{q}_{sim})\|_2^2, \end{aligned}$$

where h is the height, $\mathbf{p}$ the position and $\mathbf{q}$ the orientation of the root in world coordinates. We used $w_h = 25$, $w_{\mathbf{p}} = 15$, and $w_{\mathbf{q}} = 15$.

Pose E_{pose} measures internal joint angles to asses the overall pose:

$$E_{pose} = w_{pose} \cdot \frac{1}{N_j} \sum_{i=1}^{N_j} \|\texttt{axisAngle}(\mathbf{q}_{i,target}^{-1} \cdot \mathbf{q}_{i,sim})\|_2^2,$$

where N_j is the number of joints and $\mathbf{q}_i$ the relative orientation of the i-th joint. We used $w_{pose} = 10$.

End-Effector E_{ee} measures the motion of important body parts such as hands/feet/head/tail:

$$E_{ee} = w_{ee} \cdot \frac{1}{N_{ee}} \sum_{i=1}^{N_{ee}} \|\mathbf{p}_{i,target} - \mathbf{p}_{i,sim}\|_2^2,$$

where N_{ee} is the number of end-effectors and $\mathbf{p}_i$ is the world position of the i-th end-effector. We used $w_{ee} = 50$.

5.4. Selecting Samples

Part of our global optimization strategy is to keep a subset of the window samples and investigate the outcome at the end of the full motion. To select this subset of samples, we follow the same scheme as in [LYvdP*10], which we detail here for closure. We want to keep samples that span a diverse range of costs, while still favouring low values. To avoid picking very bad samples, which are not promising, we sort the samples by cost and discard the worst 40%. Next we map uniformly sampled values in $[0,1]$ into the range from the lowest cost c_{min} to the highest cost c_{max} of the remaining samples. Specifically, we compute the values y_i in the range $[c_{min}, c_{max}]$ as:

$$y_i = c_{min} + (c_{max} - c_{min}) \cdot \left(\frac{i}{k-1}\right)^6, \text{ with } i = 0 \,..\, (k-1),$$

where k is the number of samples to keep. For each y_i we then select the sample with the closest cost without replacement. An example of this selection process and the samples of the final control sequence is shown in Figure 7.

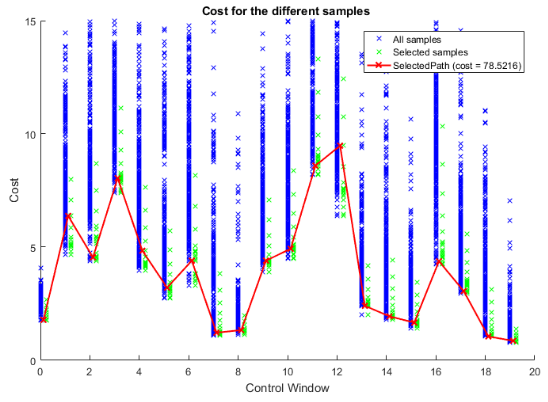

Figure 7: *The cost distribution of the generated samples (blue), the samples kept alive (green), and the final minimal cost solution (red solid line). The final solution is locally usally not the best solution, which shows the importance of keeping multiple samples alive.*

6. Results

To demonstrate the capabilities of our wide band trajectory optimization, we applied it to various motions: a walk cycle repeated multiple times, a boxing motion sequence that includes punching, blocking, as well as a hovering jumping motion. We compare with the greedy version (sequentially optimizing windows without keeping multiple samples alive) for the jumping motion, which results in a failure (see accompanying video). While we optimize for open-loop controls, the now feasible motions and controls can be used to facilitate training a closed-loop feedback policy. To demonstrate this, we trained a factored linear function that adds robust control to walking cycles, allowing the character to repeat multiple cycles.

For all our results we used the *Open Dynamics Engine (ODE) v0.15* [Smi17], with a simulation time step of 0.0005s, a friction coefficient of 0.8, and a restitution coefficient of 0.2. We used the ARBS shown in Figure 2 for each result. It was created with our modeler (see supplementary video), and holds 21 joints with a total mass of 50kg. The PD stiffness and damping coefficients are set to $k_p = 500$ and $k_d = 50$, and the torque limit is set to $\tau_{limit} = 400$.

Multiple Walk Cycles The walk cycle designed by an artist assumes the character is already moving at the first frame. In other words, it has significant momentum, and thus starting from the first frame without any initial velocities is not mechanically feasible. Therefore the trajectory optimization has to find controls that first build up some momentum, which requires a large deviation from the original motion. One can see in Figure 8 how the first cycle of the walk deviates more from the target motion, and how the character then catches up to the desired overall position and motion during the next walk cycles.

Boxing and Jumping Sequence The second motion is less structured and more diverse. It contains fast blocking and boxing movements, as well as a mechanically challenging forward jump that hovers above the ground. The blocking and boxing motions can be tracked quite accurately, without the character throwing himself out of balance, but the jumping motion is hard to recreate, as can be seen in Figure 9. Despite deviating from the original motion, our wide band trajectory optimization is robust enough to perform a similar jumping motion, while maintaining balance.

Wide Band STO Performance While we consider multiple optimizations per window, the different instances can be run in parallel. Each instance can be further parallelized by evaluating the samples in each step in independent simulations. We then run the parallelized optimizer on a cluster with 120 cores. We empirically found that generating $n = 400$ samples and keeping $k = 20$ samples alive worked well for our experiments. Given this configuration, optimizing the controls for 1 second of motion for our cartoon dinosaur requires about 25 minutes of computation. The motion with multiple walk cycles (6s) took about 2.5h, and the boxing and jumping sequence (9s) took about 4h. The bottleneck remains the simulation. It could be interesting to investigate using a fast pseudo-physics engine in early stages to explore more solutions efficiently, and then switch to a full simulator with more accurate contacts.

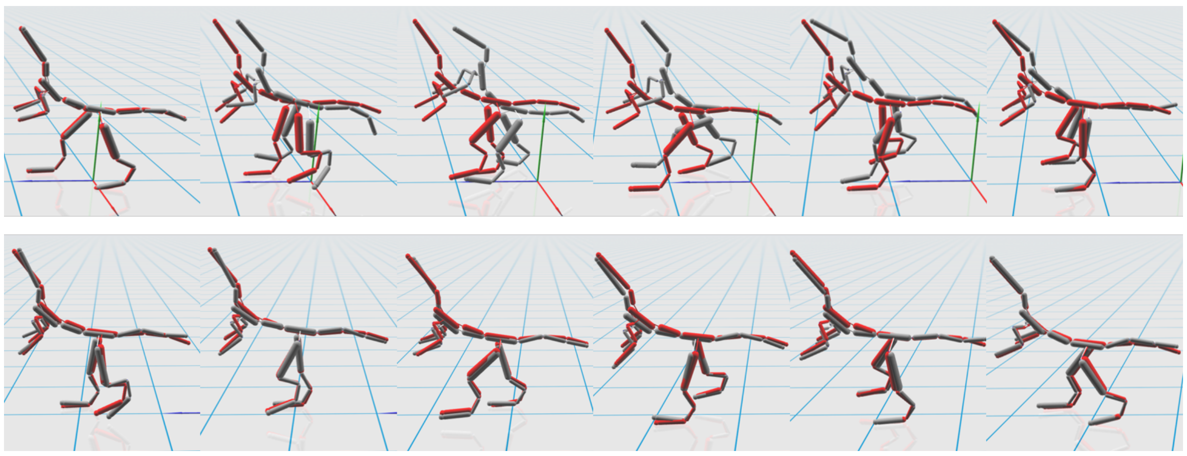

Figure 8: *The designed walk cycle assumes some initial velocity and thus starting from idle is not meachanically feasible. The optimized controls first have to build up some momentum, which leads to a large deviation from the original motion for the first cycle (top), while subsequent walk cycles can then be tracked more accurately (bottom).*

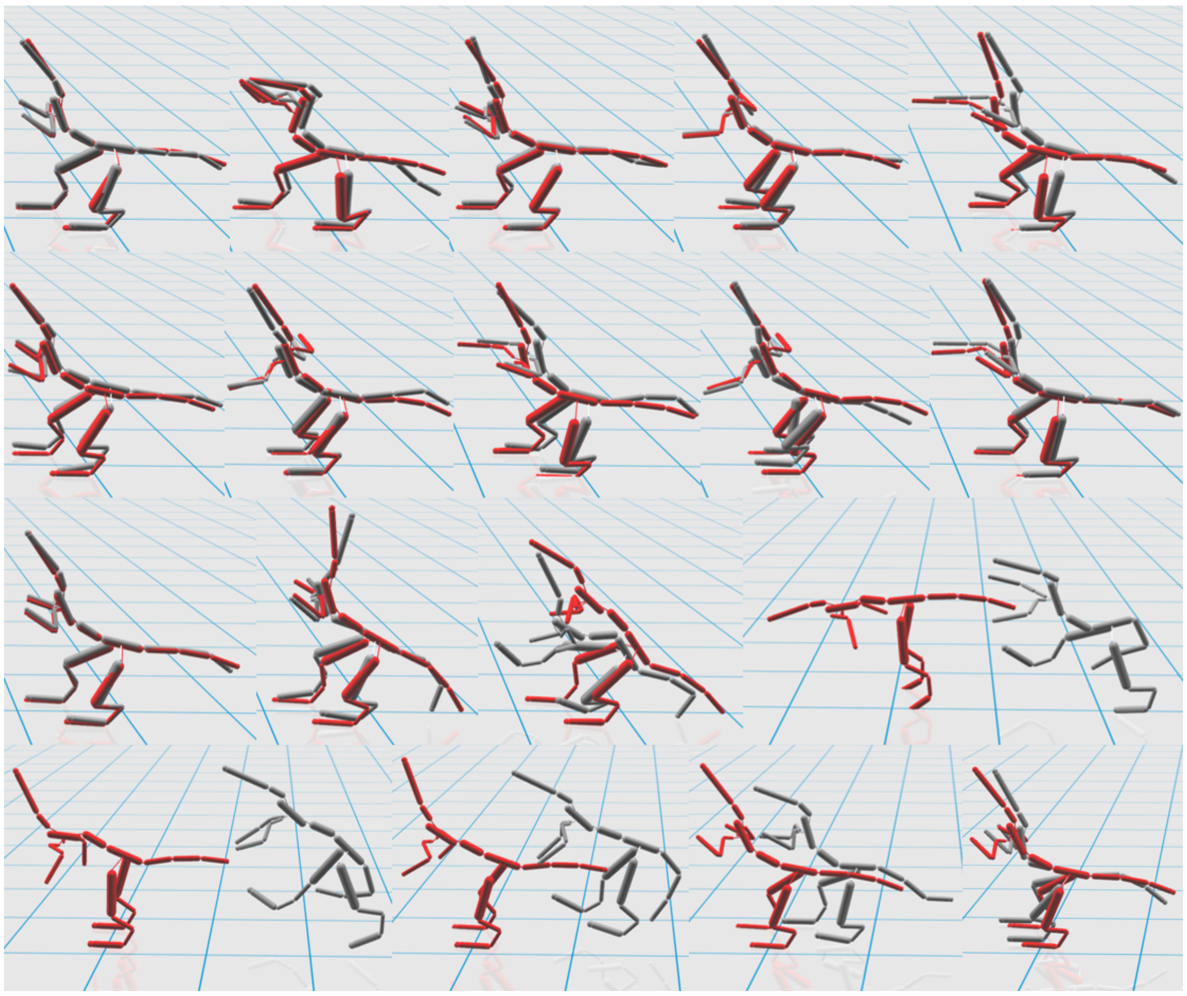

Figure 9: *The designed motion contains some fast blocking and boxing movements and a mechanically challenging forward jumping motion. While the blocking and boxing movements can be tracked quite accurately, the jumping motion requires a very large deviation from the original motion in order to perform a similar jump.*

6.1. Closed-loop Feedback Policy Example of Walking

Compared to open-loop controls, a state-dependent feedback policy allows to repeat a motion multiple times and react to slight external disturbances or perturbations. However, optimizing a state-dependant control policy directly from an infeasible motion example is very challenging, as the action space needs to cover many more possibilities. Given open-loop controls optimized with our wide band stochastic trajectory optimization, we can add a simple reduced (factored) linear feedback function to perform tasks such as walking, as shown in our supplementary video. We defer the details of the feedback policy to the appendix in the supplementary material.

7. Limitations and Future Work

While our wide band stochastic trajectory optimizer allows to transfer hand-crafted key-framed animations to simulated figures, the resulting motions are considerably noisy. We have experimented with smoothness costs, and smoother action space representations (such as low-dimensional splines), but could not find a balance between succeeding at reproducing the motions, and obtaining a smooth motion with these schemes. We believe additional factors such as the mechanical system of the character (mass distribution, joint types and collision primitive types), as well as the type of motions that we considered prevent us from investigating our method with full clarity and in a reasonable amount of time.

Control techniques are often presented along a mechanical model that has advantageous properties. Often characters will be very light, or the mass will be concentrated near the pelvis. In our experiments we kept the mass of the character proportional to its body parts, resulting in a dinosaur with a heavy head (and oscilliating head motion).

Additionally, we did not use joint-specific PD-coefficients, nor joint-specific angle search spaces (search interval). In consequence, the large stiffness (and angle) required to move large bodies, lead to large torques for light bodies.

We observed that cyclic motions are in fact more stable than momentum gaining or losing systems. A cyclic motion such as a run turns out to be easier to control than a standing still motion after stopping a run, or starting a run from an idle position. We observed that the cyclic motions were less noisy and converged faster.

Finally, another aspect of our approach that could be improved is the performance. While our sampling scheme is parallelized and adapts the sample distribution in each window, it does not share information across each optimization sub-routine. As a result, bad samples can be re-evaluated through costly simulations multiple times. One idea could be utilize a simple classifier that keeps track of bad samples and can discard early newly drawn samples instead of evaluating the cost function each time.

8. Conclusion

We presented an approach for transferring hand-crafted key-framed animations to mechanically feasible movements for under-actuated mobile figures (in a simulated environment). The motion is broken down into control windows, each optimized separately. However, instead of proceeding solely in a greedy fashion, sub-optimal control samples in the short run are kept alive throughout the global optimization, to finally recover the global optimum via backtracking. We showed examples of motions where the greedy sampling-based approach fails, compared to our wide band version. In the future, we would like to investigate ways to learn a closed-loop feedback policy using the open-loop control torques computed with our method.

References

[ABdLH13] AL BORNO M., DE LASA M., HERTZMANN A.: Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (August 2013), 1405–1414.

[AdSP07] ABE Y., DA SILVA M., POPOVIĆ J.: Multiobjective control with frictional contacts. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), Eurographics Association, pp. 249–258.

[CBvdP10] COROS S., BEAUDOIN P., VAN DE PANNE M.: Generalized biped walking control. *ACM Trans. Graph. 29*, 4 (July 2010), 130:1–130:9.

[CKJ*11] COROS S., KARPATHY A., JONES B., REVERET L., VAN DE PANNE M.: Locomotion skills for simulated quadrupeds. *ACM Trans. Graph. 30*, 4 (July 2011), 59:1–59:12.

[dLMH10] DE LASA M., MORDATCH I., HERTZMANN A.: Feature-based locomotion controllers. *ACM Trans. Graph. 29*, 4 (July 2010), 131:1–131:10.

[DLvdPY15] DING K., LIU L., VAN DE PANNE M., YIN K.: Learning reduced-order feedback policies for motion skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2015), ACM, pp. 83–92.

[GP12] GEIJTENBEEK T., PRONOST N.: Interactive character animation using simulated physics: A state-of-the-art review. *Comput. Graph. Forum 31*, 8 (Dec. 2012), 2492–2515.

[Han06] HANSEN N.: *The CMA Evolution Strategy: A Comparing Review*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 75–102.

[HET*14] HÄMÄLÄINEN P., ERIKSSON S., TANSKANEN E., KYRKI V., LEHTINEN J.: Online motion synthesis using sequential monte carlo. *ACM Trans. Graph. 33*, 4 (July 2014), 51:1–51:12.

[HL14] HA S., LIU C. K.: Iterative training of dynamic skills inspired by human coaching techniques. *ACM Trans. Graph. 34*, 1 (Dec. 2014), 1:1–1:11.

[HRL15] HÄMÄLÄINEN P., RAJAMÄKI J., LIU C. K.: Online control of simulated humanoids using particle belief propagation. *ACM Trans. Graph. 34*, 4 (July 2015), 81:1–81:13.

[HWBO95] HODGINS J. K., WOOTEN W. L., BROGAN D. C., O'BRIEN J. F.: Animating human athletics. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), SIGGRAPH '95, pp. 71–78.

[HYL12] HA S., YE Y., LIU C. K.: Falling and landing motion control for character animation. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 155:1–155:9.

[LH17] LIU L., HODGINS J.: Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Trans. Graph. 36*, 3 (June 2017), 29:1–29:14.

[LKL10] LEE Y., KIM S., LEE J.: Data-driven biped control. *ACM Trans. Graph. 29*, 4 (July 2010), 129:1–129:8.

[LvdPY16] LIU L., VAN DE PANNE M., YIN K.: Guided learning of control graphs for physics-based characters. *ACM Transactions on Graphics 35*, 3 (2016).

[LWH*12] LEVINE S., WANG J. M., HARAUX A., POPOVIĆ Z., KOLTUN V.: Continuous character control with low-dimensional embeddings. *ACM Trans. Graph. 31*, 4 (July 2012), 28:1–28:10.

[LYG15] LIU L., YIN K., GUO B.: Improving sampling-based motion control. *Comput. Graph. Forum 34*, 2 (2015), 415–423.

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

[LYvdPG12] LIU L., YIN K., VAN DE PANNE M., GUO B.: Terrain runner: Control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 154:1–154:10.

[MdLH10] MORDATCH I., DE LASA M., HERTZMANN A.: Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics 29*, 4 (2010), 1.

[MTP12] MORDATCH I., TODOROV E., POPOVIĆ Z.: Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph. 31*, 4 (July 2012), 43:1–43:8.

[MZS09] MACCHIETTO A., ZORDAN V., SHELTON C. R.: Momentum control for balance. *ACM Trans. Graph. 28*, 3 (July 2009), 80:1–80:8.

[NRH17] NADERI K., RAJAMÄKI J., HÄMÄLÄINEN P.: Discovering and synthesizing humanoid climbing movements. *ACM Trans. Graph. 36*, 4 (July 2017), 43:1–43:11.

[PBvdP15] PENG X. B., BERSETH G., VAN DE PANNE M.: Dynamic terrain traversal skills using reinforcement learning. *ACM Trans. Graph. 34*, 4 (July 2015), 80:1–80:11.

[PBvdP16] PENG X. B., BERSETH G., VAN DE PANNE M.: Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Trans. Graph. 35*, 4 (July 2016), 81:1–81:12.

[PBYvdP17] PENG X. B., BERSETH G., YIN K., VAN DE PANNE M.: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics 36*, 4 (2017).

[RH17] RAJAMÄKI J., HÄMÄLÄINEN P.: Augmenting sampling based controllers with machine learning. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2017), ACM, pp. 11:1–11:9.

[RvdPK14] RABBANI A. H., VAN DE PANNE M., KRY P. G.: Anticipatory balance control. In *Proceedings of the Seventh International Conference on Motion in Games* (2014), MIG '14, pp. 71–76.

[Smi17] SMITH R.: Open dynamics engine (ode) 0.15. Available online at `http://www.ode.org/`, 2017. Accessed on 01.01.2018.

[TGLT14] TAN J., GU Y., LIU C. K., TURK G.: Learning bicycle stunts. *ACM Trans. Graph. 33*, 4 (July 2014), 50:1–50:12.

[TGTL11] TAN J., GU Y., TURK G., LIU C. K.: Articulated Swimming Creatures. *ACM Trans. Graph. Article 30*, 10 (2011).

[WFH09] WANG J. M., FLEET D. J., HERTZMANN A.: Optimizing walking controllers. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 168:1–168:8.

[WK88] WITKIN A., KASS M.: Spacetime constraints. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 159–168.

[YL10] YE Y., LIU C. K.: Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph. 29*, 4 (July 2010), 74:1–74:9.

[YLvdP07] YIN K., LOKEN K., VAN DE PANNE M.: Simbicon: Simple biped locomotion control. *ACM Trans. Graph. 26*, 3 (July 2007).