Domain of Attraction Expansion for Physics-Based Character Control

Mazen Al Borno

University of Toronto

and

Michiel van de Panne

University of British Columbia

and

Eugene Fiume

University of Toronto

Determining effective control strategies and solutions for high degree-of-freedom humanoid characters has been a difficult, ongoing problem. A controller is only valid for a certain set of states of the character, known as the *domain of attraction* (DOA). This paper shows how states that are initially outside the DOA can be brought inside it. Our first contribution is to show how DOA expansion can be performed for a high-dimensional simulated character. Our second contribution is to present an algorithm that efficiently increases the DOA using random trees that provide denser coverage than the trees produced by typical sampling-based motion planning algorithms. The trees are constructed offline, but can be queried fast enough for near real-time control. We show the effect of DOA expansion on getting up, crouch-to-stand, jumping, and standing-twist controllers. We also show how DOA expansion can be used to connect controllers together.

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

Additional Key Words and Phrases: Computer Animation, Physics Simulation, Character Animation

ACM Reference Format:

Corresponding author: mazen@dgp.toronto.edu

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© YYY ACM 0730-0301/YYY/15-ARTXXX \$10.00

DOI 10.1145/XXXXXXX.YYYYYYY

<http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY>

1. INTRODUCTION

It is very difficult to find control solutions for the locomotion of high degree-of-freedom humanoid characters. Such solutions form small, elusive manifolds within a highly nonlinear motion domain. What's worse is that they often lack robustness (e.g., fall over with the smallest push), and are difficult to sequence into composite motions. As such, they may demonstrate the existence of control solutions but not feasibility or practicality. Given a character with skills, e.g., balancing, jumping, rising, etc., a certain set of initial states can achieve a set of desirable states. We refer to the set of initial states as the *domain of attraction* (DOA) and the set of desirable states as the *goal set*. Skills that are performed from states outside the DOA do not achieve a desirable outcome. The paper deals with the following question: how can skills be performed over a much larger set of situations? In other words, how can we expand the DOA?

Inspired by Tedrake [Tedrake 2009], we present an algorithm to do this using random trees. One contribution of this paper is to show how DOA expansion can be performed for a 25 degrees-of-freedom (underactuated) character as opposed to a 1 degree-of-freedom (fully actuated) pendulum. Secondly, we identify a major source of inefficiency in performing DOA expansion with typical sampling-based motion planning algorithms such as Rapidly Exploring Random Trees (RRTs) [LaValle 1998]. In this work, we advocate for a new motion planning algorithm that constructs *dense* random trees. This is achieved by biasing our trees to grow in breath before depth, while RRTs are biased conversely. The key intuition is that dense trees make it easier to steer a state inside the DOA.

The goal of DOA expansion is to cover as large a portion as possible of a given domain of interest, i.e., to enlarge as much as is required the set of states that can be brought to the goal set. As in Tedrake's work, we trade the aim of finding optimal policies, which would be intractable, for the aim of finding policies that are good enough, i.e., locally optimal. We perform DOA expansion on getting up, crouch-to-stand, jumping and standing-twist controllers. Our controllers are obtained by optimizing a time-indexed spline that provides proportional-derivative (PD) target angles to achieve a final target state. We could, however, have chosen to perform DOA expansion on any initial controller, e.g., [Coros et al. 2010; Abe et al. 2007]. The trees are constructed offline, but can be queried quickly enough for near real-time control. The queries use the character's

state to locate the appropriate time-indexed target PD-reference trajectory for control.

2. RELATED WORK

Physics-based motion can be synthesized by optimizing a trajectory [Witkin and Kass 1988], [Mordatch et al. 2012], [Al Borno et al. 2013] or by designing control laws for specific movements [Yin et al. 2007], [Hodgins et al. 1995], [Al Borno et al. 2014]. Open-loop controllers have small DOAs; closed-loop controllers such as SIMBICON can have large DOAs. Both are only suitable for initial states inside the DOA. We are interested in the question of how to control a simulated character outside the DOA.

There are two general approaches for character control. One is model predictive control (MPC), which re-plans a new trajectory at each timestep from the current state. The other is to pre-compute a control policy. The latter requires examining a domain in state space and preparing actions for every state that could be encountered. The former does not, making it attractive because the domain can be very large. This comes at the cost of more easily falling into local minima.

Recent work by Tassa et al. [2012] and Hämäläinen et al. [2014] are examples of MPC methods. The iLQG method of Tassa et al. can have the character get up from arbitrary lying positions on the ground. The character, however, gets up unrealistically with one bounce, implying that very large torques are at play. In our own experiments, the character fails to get up with the iLQG method when more conservative torques limits are used, e.g., ± 300 Nm. The multimodal sampling method of Hämäläinen et al. can have the character balance and get up from a wide variety of scenarios, but it fails on others. It is for the failure cases that some offline pre-computation is necessary. We discuss our results in relation to these methods in Sec. 6.4.

Our work fits in the pre-computation of a control policy category. Wang et al. [2010] optimize the parameters of single controller given a distribution of initial states, but the method does not generalize well when the initial states are too far away. Atkeson et al. [2009] demonstrate on low-dimensional problems how to approximate the globally optimal policy with local estimates of the value function. To make the problem more tractable, Tedrake [Tedrake 2009] forgoes the goal of finding globally optimal policies for the goal of finding good-enough policies instead. The main idea is to use RRTs and feedback controllers to cover a desired domain in state space. Trajectory optimization (TOPT) is used to connect two states together. Mordatch et al. [2012] and Al Borno et al. [2013] are recent TOPT methods for offline motion synthesis. Our work is inspired by Tedrake's approach. The LQR-Trees algorithm uses direct collocation to synthesize open-loop trajectories, the linear quadratic regulator (LQR) for feedback control, and Lyapunov functions for estimating the DOA. Our work uses the model-free, shooting method of Al Borno et al. [2013] to synthesize trajectories stabilized with PD controllers indexed by time, and forward dynamics to evaluate the DOA. The method provides a fundamentally different approach to the single-optimized-trajectories that result from MPC; it develops a tree of reference trajectories that is computed offline and then exploits it online. While previous character animation papers have produced workable DOAs in some scenarios, none of them answer the question of how to significantly grow the DOA.

We perform DOA expansion on crouch-to-stand, jumping, standing-twist, and various getting up controllers. There is previous work on getting up motions by Lin et al. [2012], where motion capture data of the character getting up from one initial state is extended to new lying postures by using physics simulation. Our synthesized motions are entirely physically based, requiring no motion capture stitching. An important and under-studied problem in character animation and robotics is how to connect controllers together, i.e., how can a character perform skills sequentially, like getting up, followed by a standing-twist and a jump? In related work, Liu et al. [2012a] uses policy search to learn linear feedback matrices for very specific motion transitions. Firmin et al. [2015] provides a control language for skill authoring, where the resulting DOA depends on the skill of the designer. None of these approaches are as generic as the one we develop here. We show how DOA expansion can be used to connect controllers together.

3. DOMAIN OF ATTRACTION EXPANSION

We now present the DOA expansion algorithms. Let Γ be a tree of tuples $(\mathbf{x}, \mathbf{C}, p, T)$, where $\mathbf{x}$ is a state, $\mathbf{C}$ is a feedback controller, p is a pointer to the parent node, and T is the duration of the edge. Let χ and Ω denote the current and desired DOAs. We use the term $\mathbf{x}^{target}$ to refer to a state inside the DOA of an initial controller $\mathbf{C}_0$. A state is inside the DOA of a controller if, using the controller from that state, the character's end state is inside a goal set, i.e., within an epsilon distance of a goal state $\mathbf{x}^{goal}$. Let T_0 denote the duration required to bring $\mathbf{x}^{target}$ to the goal set using $\mathbf{C}_0$. In this work, we perform DOA expansion on initial controllers to make them suitable for a larger set of initial states.

Algorithm 1 RRTFC algorithm

```

1:  $\Gamma.add\_node(\mathbf{x}^{goal}, \text{NULL}, \text{NULL}, 0)$ 
2:  $p \leftarrow$  pointer to the root
3:  $\Gamma.add\_node(\mathbf{x}^{target}, \mathbf{C}_0, p, T_0)$ 
4: for  $k = 0, \dots, K$  do
5:   Randomly sample a state  $\mathbf{x}^{rand}$  inside  $\Omega$ 
6:   if  $\mathbf{x}^{rand} \notin \chi$  then
7:     Find the nearest neighbor  $\mathbf{x}^{near}$  in  $\Gamma$  to  $\mathbf{x}^{rand}$ 
8:     Obtain state  $\mathbf{x}^{new}$  by extending  $\mathbf{x}^{rand}$  towards  $\mathbf{x}^{near}$ 
9:     Solve a TOPT to steer  $\mathbf{x}^{new}$  to  $\mathbf{x}^{near}$ 
10:     $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T \leftarrow$  full trajectory from the TOPT
11:    if  $\mathbf{x}_T \in \chi$  then
12:      Obtain a feedback controller  $\mathbf{C}$ 
13:       $p \leftarrow$  pointer to the node that has  $\mathbf{x}_T$  in its DOA
14:       $\Gamma.add\_node(\mathbf{x}_1, \mathbf{C}, p, T)$ 
15:       $\Gamma.add\_node(\mathbf{x}_2, \mathbf{C}, p, T - 1)$ 
16:      ...
17:       $\Gamma.add\_node(\mathbf{x}_{T-1}, \mathbf{C}, p, 1)$ 
return  $\Gamma$ 

```

3.1 RRTFC Algorithm

We begin by presenting the main algorithm in Tedrake's work, which we call the RRT Feedback Coverage (RRTFC) algorithm (see Algorithm 1). The main property of RRTs is to bias the trees towards the unexplored regions in the domain. This is achieved by sampling a random state inside Ω before choosing a state $\mathbf{x}^{new}$ inside the same domain, but closer to the tree (see Fig. 1). A naive random tree algorithm would have directly sampled a random state

near the tree. As shown in Lavalle et al. [2001], the RRT is remarkably more efficient at finding a path to a target than the naive random tree.

A trajectory optimization (TOPT) problem is then solved to steer $\mathbf{x}^{new}$ towards its closest state $\mathbf{x}^{near}$ in the tree. For high-dimensional, highly-nonlinear problems, the TOPT often fails to steer $\mathbf{x}^{new}$ close enough to $\mathbf{x}^{near}$ to be inside the DOA. In our experience, this is a major source of inefficiency in performing DOA expansion because solving TOPT problems is time-consuming. Note that the connection is successful if the state at the end of the trajectory $\mathbf{x}_T$ is inside the DOA of any controller in the tree, not necessarily the DOA of the controller associated with $\mathbf{x}^{near}$ (see Fig. 2). We do not test if the states at the previous timesteps are inside the DOA because it would be too expensive. If the connection is successful, the states at all timesteps are added to the tree, even if they are not inside Ω . This increases the likelihood of $\mathbf{x}_T$ landing inside the DOA.

Fig. 1: DOA expansion with an RRT. An RRT starts by sampling a random state in the domain $\mathbf{x}^{rand}$ and finds its closest node $\mathbf{x}^{near}$ in the tree. It then attempts to connect to $\mathbf{x}^{near}$ (green arrow) starting from a state which is close and in the direction of $\mathbf{x}^{rand}$. The intuition behind DFC is that attempting to connect to the dense parts of the tree is more likely to be successful (orange arrow) even if it is slightly further away according to the distance metric. The red and purple circles are the target and goal states, and the arrow in between is the initial controller C_0 . The blue background is the domain covered by the tree.

3.2 DFC Algorithm

RRTs have the property of growing long branches quickly to efficiently explore the domain and find a path to a target state. This property does not seem important in the context of DOA expansion, where the objective is to find a path to every state in domain as opposed to a single target state. This suggests that we can design a motion planning algorithm specifically to improve the efficiency of the DOA expansion process. The algorithm should aim to construct a dense tree since this increases the likelihood of $\mathbf{x}_T$ being inside the DOA, thereby reducing the number of failed connections (see Fig. 1). We now present our approach to do this, which we call the Dense Feedback Coverage (DFC) algorithm (see Algorithm 2).

The idea is to cover the domain around $\mathbf{x}^{target}$ with progressively larger subsets $\Omega_0 \subset \Omega_1 \subset \dots \subset \Omega$, where Ω_0 denotes the DOA

Fig. 2: Examples of Trajectory Optimization in a Sparse and Dense Tree. The blue tubes illustrate the domain covered by edges of the tree. The orange circle is a random state that we attempt to connect to its nearest state in the tree with trajectory optimization. The dotted curves are the trajectories followed when attempting the connection. The figures show that the attempt is more likely to be successful in a dense tree (right figure) than in a sparse tree (left figure).

Algorithm 2 DFC algorithm

```

1:  $\Gamma.add\_node(\mathbf{x}^{goal}, \text{NULL}, \text{NULL}, 0)$ 
2:  $p \leftarrow$  pointer to the root
3:  $\Gamma.add\_node(\mathbf{x}^{target}, C_0, p, T_0)$ 
4: Divide the domain in sets  $\Omega_0 \subset \Omega_1 \subset \Omega_2 \subset \dots \subset \Omega_N$ , where  $\Omega_N = \Omega$ 
5: for  $i = 1, \dots, N$  do
6:   for  $k = 0, \dots, K_i$  do
7:     Randomly sample a state  $\mathbf{x}^{rand}$  inside  $\Omega_i$ 
8:     if  $\mathbf{x}^{rand} \notin \chi$  then
9:       Find the nearest neighbor  $\mathbf{x}^{near}$  in  $\Gamma \cap \Omega_h$  to  $\mathbf{x}^{rand}$ , where  $h < i$ 
10:      Solve a TOPT to steer  $\mathbf{x}^{rand}$  to  $\mathbf{x}^{near}$ 
11:       $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T \leftarrow$  full trajectory from the TOPT
12:      if  $\mathbf{x}_T \in \chi$  then
13:        Obtain a feedback controller  $C$ 
14:         $p \leftarrow$  pointer to the node that has  $\mathbf{x}_T$  in its DOA
15:         $\Gamma.add\_node(\mathbf{x}_1, C, p, T)$ 
16:         $\Gamma.add\_node(\mathbf{x}_2, C, p, T-1)$ 
17:        ...
18:       $\Gamma.add\_node(\mathbf{x}_{T-1}, C, p, 1)$ 
return  $\Gamma$ 

```

of the initial controller (see left Fig. 3 and Sec. 4.3 for an example of how to define the subsets). For each domain Ω_i , the nearest neighbor is constrained to be in a subset Ω_h , $h < i$. A connection is successful as long as the state at the end of the TOPT lands somewhere in Ω_h that is covered by the tree. This is why it is much easier to steer the character inside the DOA of a region occupied with many nodes as opposed to a region occupied with very few nodes. In DFC, each subsequent subset effectively gets a larger target area to perform new connections. This is akin to how a problem is divided and solved in dynamic programming. To solve the problem of bringing a state in a given domain inside the DOA, we use the solutions of subproblems found at decreasing subsets of the domain.

The optimal number of subsets Ω_k will depend on the problem. Generally, more subsets yields more samples that are already inside the DOA, while fewer subsets yields more samples that fail to connect. We choose the maximum number of iterations K_i to be roughly proportional to the fraction of the domain that is covered by the subset. The choice of the nearest neighbor subset (line 9) will also depend on the problem. To some extent, the smaller the

subset, the more likely it is for the final state of the TOPT to land inside the DOA. However, a subset that is too small, i.e., too far away from $\mathbf{x}^{rand}$, will increase the likelihood of a failed connection. A simple strategy is to choose the smallest subset such that the distance between $\mathbf{x}^{rand}$ and $\mathbf{x}^{near}$ is within a specified value.

If the DOA of the initial controller is particularly small, it might be inefficient to attempt to steer all the nearby states inside it. In other words, when the current DOA is very small, it could be even harder to steer the nearby states inside a subset of the DOA, which is the strategy employed by DFC. In this case, we first perform a DOA expansion on a subset of the desired domain using other methods, e.g., by using RRTFC on Ω_1 . We then continue the DOA expansion process on the entire domain with DFC.

3.2.0.1 Remarks. A sample that fails to connect to the tree at a given subset can successfully connect when resampled at the same or larger subset. The difference would result from having a new nearest neighbor or from the end state after a TOPT to be inside a region now covered by the tree. This is why we sample from Ω_i and not from $\Omega_i \setminus \Omega_{i-1}$, i.e., we do not exclude Ω_{i-1} . However, it is possible to sample as a function of the domain covered by the tree. For example, if the tree covers 95% of Ω_{i-1} , we can have 5% of the samples in Ω_{i-1} and 95% in Ω_i . Sec. 4.8 describes how we can estimate the domain coverage.

Fig. 3: (Left) DOA coverage with DFC. The DFC algorithm progressively covers the domain around the target state (the red circle). Increasingly larger subsets are illustrated in the figure. The orange circle is a random state sampled on the largest subset. TOPT is used to steer the state to its closest state in a smaller subset. **(Right) Determining if a state is inside the DOA.** We illustrate how we determine if a state $\mathbf{x}^r$ (the orange circle) is inside the DOA. In the figure, we perform simulation rollouts, starting from the controllers associated with the closest states in the tree (the orange arrows). A rollout consists of moving up the nodes of a branch, until the root (the goal state) is reached. If one of the rollouts steer the character close enough to the goal state, then $\mathbf{x}^r$ is inside the DOA.

4. TECHNIQUES

We now describe our implementations of the operations in the RRTFC and DFC algorithms.

4.1 Trajectory Optimization and Feedback Control

The method of Al Borno et al. [2013] optimizes a reference trajectory $\hat{\mathbf{q}}_{1:T}$ represented by an arc-length parametrized cubic B-spline. The output motion $\mathbf{q}_{1:T}$ is computed by forward dynamics.

The torque u_t for a joint is determined by PD control:

$$u_t = k_p(\hat{q}_t - q_t) - k_d\dot{q}_t, \quad (1)$$

where q_t and $\hat{q}_t$ are the current and reference values of the joint angle. All joints have PD gain values of $k_p = 700 \text{ Nm/rad}$ and $k_d = 1 \text{ Nms/rad}$.

We use this TOPT method to steer the character to a desired state $\mathbf{x}^d = (\mathbf{q}^d, \dot{\mathbf{q}}^d)$ in T timesteps, while minimizing a measure of effort. The optimization problem is given by:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} w_1 \text{dist}(\mathbf{x}_T, \mathbf{x}^d) + w_2 \sum_{i=1}^T \|\mathbf{u}_i\|^2, \quad (2)$$

where $\mathbf{s}$ is the vector of spline knots, the distance metric dist is defined in Sec. 4.4, $\mathbf{u}$ is the vector of joint torques, and w_1 and w_2 are weights. The optimization is performed with CMA [Hansen 2006] and is initialized with a kinematic interpolation between the start and desired poses. The spline knots are spaced every 0.1s.

4.1.0.2 Remarks. Note that this method returns both the trajectory and the closed-loop controller in the RRTFC (lines 9 and 12) and DFC (lines 10 and 13) algorithms. Although our joint-local PD controllers are weak, i.e., have small DOAs, we can build robust controllers by combining many weak ones together. Alternatively, one could optimize these controllers to make them more robust, e.g., as in Liu et al. [2012b]. There is a trade-off on whether it is more efficient to spend time building a few robust controllers or combining many weak ones.

A simple extension to the method of Al Borno et al. [2013] can be done to optimize for the duration of the movement. We add a time variable to the optimization and a cost of time objective to penalize excessively slow movements. Specifically, we add a linear cost $w_3 T$ to Eq. (2). Optimizing the movement duration every time the algorithm attempts to connect two states would be prohibitively slow. For this reason, whenever the distance between the initial and desired state is less than a chosen threshold, we use a constant movement duration, e.g., 0.2s.

4.2 DOA modeling

We now describe how we determine if a state $\mathbf{x}^r$ is inside the DOA after a TOPT, which corresponds to line 11 in RRTFC and line 12 in DFC. We model the DOA in an implicit fashion by using multiple forward dynamics simulations, rather than having an explicit model as in Tedrake [Tedrake 2009], which currently does not scale to high dimensions. We start by finding the V closest states to $\mathbf{x}^r$. We then perform V simulation rollouts starting from $\mathbf{x}^r$, using the controllers that were optimized in Sec. 4.1. The rollouts proceed until the root of the tree is reached (see right Fig. 3). We then compare the V final states with the goal state. If at least one of the states is within an epsilon ball of the goal state, then we consider $\mathbf{x}^r$ to be inside the DOA. Pseudo-code is provided in Algorithm 3. The reason we consider V closest states instead of the closest state only is that any distance metric will be imperfect. We often find $\mathbf{x}^r$ to be inside the DOA of a different node than the “closest” node. We usually choose V to be between 50 and 250. The larger the value, the more likely it is that we find a path that leads to the goal set, the more time-consuming the DOA expansion process becomes.

4.2.1 Minimal Time. Note that Algorithm 3 halts when the first of V rollouts leads to the goal set. One could continue performing

the remaining rollouts to not only find a path to the goal set, but a path that minimizes some objective (time, energy, etc). Algorithm 4 minimizes the time objective. While this slightly slows down the DOA expansion process, it quite noticeably improves the quality of the motions, as shown in the accompanying video.

4.2.2 Using the Tree Online. Once the tree is constructed, Algorithm 3 can be used online to control the character. When searching for the nearest nodes in the tree, we only take those at timestep 1 (line 15 in Algorithm 2) into consideration to speed-up the process because we know that they are inside the desired domain Ω . This is also how we determine if a randomly sampled state is inside the DOA in the tree construction process, which corresponds to line 6 in RRTFC and line 8 in DFC. On a single core, Algorithm 3 runs on average in real-time because only a few rollouts are usually required before finding a solution. The rollouts should not be long for this method to achieve interactive rates, e.g., 4 s. We have tried learning a nearest neighbor classifier that maps a state to a controller, i.e., a branch in the tree, to bypass the need to perform the rollouts. This required an excessively high number of samples for a small domain, making it poorly scalable.

Algorithm 3 [IsInside, Parent] = findStartState($\mathbf{x}^r$)

```

1: IsInside  $\leftarrow$  false
2: Parent  $\leftarrow$  NULL
3: Find the  $V$  closest nodes to  $\mathbf{x}^r$ 
4: for  $i = 1, \dots, V$  do
5:    $n \leftarrow$  pointer to the  $i$ -th closest node
6:    $\mathbf{x}^f \leftarrow \mathbf{x}^r$ 
7:   while  $n \neq \text{ROOT}$  do
8:     Do a rollout from  $\mathbf{x}^f$  with the  $n_C$  controller
9:      $\mathbf{x}^f \leftarrow$  state at the end of the rollout
10:     $n \leftarrow n_p$  (pointer to the parent node)
11:   if  $\text{dist}(\mathbf{x}^f, \mathbf{x}^{\text{goal}}) < \epsilon$  then
12:     IsInside  $\leftarrow$  true
13:     Parent  $\leftarrow$  pointer to the  $i$ -th closest node
14:   break
```

Algorithm 4 [IsInside, Parent] = findBestStartState($\mathbf{x}^r$)

```

1: IsInside  $\leftarrow$  false
2: Parent  $\leftarrow$  NULL
3:  $s \leftarrow \infty$ 
4: Find the  $V$  closest nodes to  $\mathbf{x}^r$ 
5: for  $i = 1, \dots, V$  do
6:    $t \leftarrow 0$ 
7:    $n \leftarrow$  pointer to the  $i$ -th closest node to  $\mathbf{x}^r$ 
8:    $\mathbf{x}^f \leftarrow \mathbf{x}^r$ 
9:   while  $n \neq \text{ROOT}$  do
10:    Do a rollout from  $\mathbf{x}^f$  with the  $n_C$  controller
11:     $\mathbf{x}^f \leftarrow$  state at the end of the rollout
12:     $n \leftarrow n_p$ 
13:     $t \leftarrow t + n_T$ 
14:   if  $\text{dist}(\mathbf{x}^f, \mathbf{x}^{\text{goal}}) < \epsilon$  then
15:     IsInside  $\leftarrow$  true
16:     if  $t < s$  then
17:        $s \leftarrow t$ 
18:     Parent  $\leftarrow$  pointer to the  $i$ -th closest node
```

4.3 Sampling States

The DFC algorithm samples states in the progressively larger domains $\Omega_1, \dots, \Omega_N$. This is performed as follows for the getting up motions of the simulated character. We parametrize each actuated joint in Euler angles and treat each axis independently for multi-dimensional joints. For joint j with pose r_j in the target state, the pose of the generated state is sampled uniformly in the interval $[\max(r_j - \alpha_i/m_j, l_j), \max(r_j + \alpha_i/m_j, u_j)]$, where α_i is a scalar that parametrizes Ω_i , m_j is the mass of the associated body link, l_j and u_j are lower and upper joint limits. The Euler angle parametrization can lead to a non-uniform sampling of rotations due to the singular configurations [Kuffner 2004]. Our joint limits, however, avoid this issue. The root orientation of the character is parametrized with a quaternion. The orientation of the sampled state is determined by spherical linear interpolation of a random orientation and the target state orientation, where the interpolation parameter is a function of α_i . We then drop the character from the air and wait a short amount of time. The state of the character lying on the ground is our random sample. For the other controllers, the state is sampled as follows. Starting from the target state, we apply an external force in a random direction on the character for a duration of 0.1s. The random sample is the state immediately after the external force is applied. The magnitude of the force is chosen randomly on the interval $[0, \kappa_i]$, where the scalar κ_i parametrizes Ω_i . The values of α_i and κ_i increase with i .

4.4 State Distance Metric

Given a random state $\mathbf{x}^{\text{rand}}$, the DOA expansion algorithms need to find its nearest state in the tree. The metric used to measure the distance between states $\mathbf{x}$ and $\mathbf{x}'$ can have an important impact on the efficiency of the algorithms, but it plays an even more critical role in RRTFC because DFC is designed to reduce the importance of the nearest neighbor. For simplicity, we use the weighted joint angles metric:

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \sum_j w_j(r_j - r'_j) + \tilde{w}_j(\dot{r}_j - \dot{r}'_j), \quad (3)$$

where the sum is over the joint angles and the root angle, w_j is set to the mass of the associated body link, and $\tilde{w}_j = 0.1w_j$.

4.5 Optimality Heuristic

After sampling a random state, the choice of its nearest neighbor has an important effect on the quality of the motion. One optimality heuristic is to choose N nearest neighbors instead of one, and solve N TOPTs. Of the successful connections, we choose the one with the minimal duration to the target. This approach slows down the DOA expansion process proportionally to N . In the supplemental video, we compare the results between $N = 1$ and $N = 10$.

4.6 Connecting Controllers

An important application of DOA expansion is in connecting different controllers together. If the state at the end of one controller is close enough to DOA of the second controller, then DOA expansion can be directly applied on the second controller for the connection. The only difference in Algorithm 2 is that the samples are now chosen randomly from the distribution of the states at the

end of the first controller. If the first state is too far from the DOA, then we start by synthesizing a transition controller (see Sec. 4.1) that steers it as close as possible to the DOA. We then perform DOA expansion on the transition controller, again with the random samples taken from the distribution of states at the end of the first controller.

4.7 Hybrid of Offline and Online Optimization

The method described so far requires pre-specifying an action for every possible state that could be encountered in the desired domain, which is generally very large due to the dimensionality of the character. MPC methods, on the other hand, do not require this pre-computation, but more easily fall into local minima. We now present a hybrid approach that attempts to get the best of both worlds, i.e., combine MPC methods with *some* offline pre-computation to avoid both the curse of dimensionality and the local minima. The idea is to construct a small tree offline that sketches the general path to arrive at the goal state. Online optimization is then used to track a branch in the tree. In more details, to determine if a state $\mathbf{x}^r$ is inside the DOA (lines 8 and 12 in Algorithm 2), we first find its nearest state in the tree $\mathbf{x}^{near}$. We then perform an online trajectory optimization to steer $\mathbf{x}^r$ towards $\mathbf{x}^{goal}$. For the simulated character, our online optimization method consists of 100 CMA iterations of 30 short rollouts (e.g., 0.7 s), which runs at about 5% real-time on a machine with 8 logical cores. Fewer iterations would be required for longer rollouts or different methods could be used, e.g., [Hämäläinen et al. 2014]. The cost function used is Eq. 2, with $\mathbf{x}^d = \mathbf{x}^{goal}$. At each timestep t , the control torque is determined by $\mathbf{u}_t = \mathbf{u}_t^{near} + \gamma \mathbf{u}_t^*$, where $\mathbf{u}_t^{near}$ is the control torque associated with $\mathbf{x}^{near}$, $\mathbf{u}_t^*$ is the newly optimized control torque and γ is a weight, which we set to 0.05. We found this technique to converge faster than to simply initialize the new trajectory optimization with its nearest neighbor's solution.

4.8 DOA Volume

To estimate the proportion of the domain that is covered by the tree, we sample a large number of states inside the domain and determine if they are inside the DOA using Sec. 4.2.2. The proportion of samples inside the domain provides the estimate of the proportion of the domain that is covered.

5. PLANAR PENDULUM

We first perform DOA expansion on a simple system to compare the efficiency of RRTFC and DFC. The system is a planar pendulum in the vertical plane with state $\mathbf{x} = [\theta, \dot{\theta}]$, where θ is the joint position. The mass of the pendulum is 3.10 kg and the torque limits are ± 200 Nm. The goal is to stabilize the pendulum to its unstable equilibrium $[\pi, 0]$, i.e., the upright position, from all the initial states in the domain $\Omega = [0, 0] \times [2\pi, 0]$. We consider that the pendulum has reached the equilibrium if $\mathbf{x} \in \Omega_0$, where $\Omega_0 = \{\forall \mathbf{x} : \|\mathbf{x} - [\pi, 0]\|_2 < 0.01\}$. We solve the TOPT with the method of Sec. 4.1, where we limit the number of CMA iterations to 20 and the number of samples to 12. For the DFC algorithm, we divide the domain in 4 subsets Ω_k , defined as follows: $\Omega_k = [\pi(1 - \alpha_k), 0] \times [\pi(1 + \alpha_k), 0]$, where $\alpha_k = k/4$, and $k = 1, \dots, 4$. When using the DFC algorithm, we perform a DOA expansion on Ω_1 using RRTFC because Ω_0 is too small to be used as a target for all the nearby states.

We compare RRTFC (on the entire domain) and DFC for an average of 10 runs, where the entire domain is probabilistically covered, i.e., 200 consecutive random samples are inside the DOA. Given that determining if a state is inside the DOA is much less expensive than a TOPT, the main factors that will affect the efficiency of the algorithms are the number of samples that fail to be added to the tree n_f and the number of samples that are successfully added n_s . RRTFC has $n^f = 38$ and $n^s = 13$, while DFC has $n^f = 13$ and $n^s = 16$. In this case, DFC reduces the number of failed connections by about 65%, but requires 23% more nodes to cover the domain. Overall, DFC is more efficient since it reduces the total number of TOPTs by 44%. We obtain similar results when dividing the domain in 8 or 16 subsets. Similarly, we compared the algorithms on a double pendulum with state $\mathbf{x} = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$, where each link has mass 3.10 kg. For the TOPT, we limit the number of CMA iterations to 50 and the number of samples to 25. Covering the domain $\Omega = [\pi - 0.2, \pi - 0.2, 0, 0] \times [\pi + 0.2, \pi + 0.2, 0, 0]$ required RRTFC $n^f = 950$ and $n^s = 25$, and DFC $n^f = 132$ and $n^s = 25$, which corresponds to an 84% difference in efficiency.

6. SIMULATED CHARACTER

We perform DOA expansion on getting up, crouch-to-stand, jumping, and standing-twist controllers for a simulated character. The getting up controllers express different strategies (see Fig. 5). Our initial controllers are synthesized by manually specifying between 1 and 3 key-frames as targets in a TOPT problem (Sec. 4.1) with the last key-frame corresponding to the desired end state, and by specifying the durations between poses. The goal of DOA expansion is to increase the set of initial states from which the controllers can be used. We also regulate the final global orientation of the character, i.e., it must get up, twist, etc., at a given *facing* direction, which makes the problem harder.

6.1 Full Branch Rollouts

Performing DOA expansion on the crouch-to-stand, jumping and standing-twist controllers proved to be particularly more challenging than the getting up controllers, probably because momentum effects are more present. The initial controllers have so small DOAs that it is very difficult to steer any state inside them. For this reason, we modify the TOPT method as follows. Let $\mathbf{C}$ denote an arbitrary controller that steers the character from state $\mathbf{x}_1$ to $\mathbf{x}^{goal}$. When performing a TOPT to connect a randomly sampled state to the DOA of $\mathbf{C}$, we previously attempted to minimize the distance of the state at the end of the rollout to $\mathbf{x}_1$. Now, our TOPTs will minimize the distance to $\mathbf{x}^{goal}$, using $\mathbf{C}$ in the rollouts. This significantly slows down the DOA expansion process since we are now performing rollouts for the entire duration of the branch of the tree instead of the edge only (see Fig. 4). Once the DOA of the tree gets sufficiently large, it is no longer necessary to perform the full branch rollouts.

6.2 Facing Direction Invariance

The task of our getting up controllers is to have the character reach standing balance with a desired facing direction. Removing the facing direction constraint simplifies the problem. One way to build a rising controller with a large DOA is to supply a single prone or supine target pose for the character to achieve, which should not be difficult because of the facing direction invariance, and wait

Fig. 5: Getting Up Controllers. The figure is divided in 3 pairs of sequences. For each pair, the top sequence illustrates an initial getting up controller and the bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The yellow arrow points to the desired facing direction for the final pose of the character. The red character is the pose of the currently active node.

until the motion comes largely to rest. The target pose provides a nicely repeatable starting state from which to then apply some known rising strategy, thus avoiding the need for DOA expansion. In the accompanying video, we show that this technique does not work with our controllers. The reason is that the character never perfectly reaches the target pose. Very slight errors can make the controllers fail. Our experiments show that a small tree, e.g., 80 nodes, is sufficient to obtain a robust rising controller that is invariant with respect to the facing direction.

6.3 Implementation Details

In the TOPT step of the DFC algorithm (line 10), we use a maximum of 600 CMA iterations and 150 samples. For our simulations, we use the MuJoCo physics engine [Todorov et al. 2012] with a

0.01 s timestep, a coefficient of friction of 1, and torque limits of ± 150 Nm. Our simulated character has a mass of 40.5 Kg, a height of 1.61 m, and 25 degrees-of-freedom.

6.4 Results

The getting up, crouch-to-stand and standing-twist controllers were constructed offline, and the jumping controller was constructed with the offline-online approach (Sec. 4.7). A standing controller with a similar approach to Hämäläinen et al. [2014] is used when the character is near the last standing pose. The number of nodes required to cover a given subset depends on the controller. This can be seen in Fig. 9, where we plot the total number of nodes required to cover progressively larger subsets for our getting up controllers. Intuitively, we would expect the rate of growth of the tree to in-

Fig. 6: Crouch-to-Stand Controller. The top sequence consists of key-frames of the crouch-to-stand controller. The bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The blue line on the first frame is an external force applied on the head. The red character is the pose of the currently active node.

Fig. 7: Exploration regions. The sequence shows typical states (orange character) in progressively larger subsets for the target states (the red character) associated with the “Getting Up 1” and the crouch-to-stand controllers. The subsets are $\alpha = 1$, $\alpha = 5$, and $\alpha = 10$ for the first 3 figures, and $\kappa = 100 \text{ Nm}$, $\kappa = 400 \text{ Nm}$, and $\kappa = 1000 \text{ Nm}$ for the last 3 figures.

Fig. 4: Full Branch Rollouts. In this figure, we attempt to connect the orange node to its nearest node in the tree. The typical TOPT approach is to steer the node towards the nearest node (the dotted orange arrow). In some problems, we were not able to make any connection this way. We modify the TOPT by including the controllers on the closest branch in the rollouts (the orange arrows). The objective is to steer the character as close as possible the goal state. This optimization is more likely to succeed than attempting to find a motion directly to the goal state (the green arrow).

crease with larger subsets. The plot suggests that the trees grows exponentially with the subset parameter (the vertical axis is in log scale). In Fig. 8, we plot some of the nodes in the trees for the “Getting Up 2” and the crouch-to-stand controllers to help visualize the expansion process. Table I provides some statistics on the

Fig. 9: The Size of the Tree. The figure plots the size of the tree in log scale required to cover a subset for the “Getting Up 1”, the “Getting Up 2” and the “Getting Up 3” controllers. The subsets are parametrized by α , which is defined in Sec. 4.3. In the plot, the value of α is shown in the Subset-axis. We could have continued the expansion process for larger subsets.

trees constructed for our controllers. The trees required between 1 and 3 days of computation on a desktop computer.

Fig. 8: Visualizing the nodes in the Tree. The first 3 figures shows the DOA Tree with 1, 10 and 40 nodes for the “Getting Up 2” controller. The last 3 figures shows the DOA Tree with 1, 10 and 40 nodes for the crouch-to-stand controller. These nodes are sampled from Ω . The difference in velocities between the nodes cannot be visualized.

Performing DOA expansion on an initial motion effectively extends it to new initial states. If the goal is to get up from an arbitrary lying position, the method does not attempt to discover how to get up from scratch. Instead, it attempts to find a path to the place where a getting up strategy is given. In the accompanying video, we show that the character can get up with the desired facing direction from a large set of initial states (see Fig. 5). The strategies employed show the character moving from prone to supine (and vice versa), and rotating itself on the ground to regulate its global orientation. Similarly, it is shown how the character can successfully perform crouch-to-stand, jumping and standing-twist motions when force impulses on the order of 7.5 Ns, 10 Ns and 12.5 Ns are applied in random directions on the character and the results are contrasted when no DOA expansion is performed (see Fig. 6). The discovered solutions include the character taking a step or leaning on its knee to stay in balance and perform the motion. In contrast, the Sequential Monte Carlo (SMC) method of Hämäläinen et al. [2014] fails to have the character get up under certain prone and supine postures. The same comment applies to the method of Tassa et al. [2012] when more realistic torque limits are used. In the supplemental video, we show how the iLQG provided in MuJoCo and our implementation of the SMC method fall into local minima when attempting to steer the character inside the DOA of the standing-twist controller. For the SMC method, we implemented Algorithm 1 in Hämäläinen et al. [2014] since our problem does not have a dynamically changing environment, and we compute the torques by PD control instead of their approach, which is specific to the ODE physics simulator. DOA expansion can be used to steer the “failure” states inside the DOA of these MPC methods. The accompanying and supplemental videos contain examples of how DOA expansion can be used to connect controllers together. We show how a crouch-to-stand controller can be connected to a jumping controller, using a standing-twist transition controller. We also show how jumping and stand-crouch controllers can be used repeatedly in a sequence.

The quality of the the trajectory optimizer, i.e., how closely can it steer the character towards the target, has an important impact on the efficiency of DOA expansion. Intuitively, we expect that the less accurate the optimizer is, the more beneficial it will be to have a dense tree. We perform the following experiment to test this hypothesis. We construct 5 CMA optimizers of increasing quality that are obtained by increasing the total number of samples in the roll-outs. The optimizers are then used to perform DOA expansion on the domain $\alpha = 0.5$ for one of our getting up controllers. Table II provides the number of TOPTs ($n_s + n_f$ in Sec. 5) required to cover the domain for average of 3 runs. The duration of the roll-outs is set to 0.2s so that the TOPTs are equally costly. The data indicates that DFC is more computationally efficient than RRTFC, particularly when it is difficult to connect two states together with TOPT.

Controller	Size	Height	Depth	Subset
Getting Up 1	1377	14	7.3	$\alpha = 8$
Getting Up 2	166	6	2.9	$\alpha = 3$
Getting Up 3	751	12	5.4	$\alpha = 8.5$
Crouch-to-stand	193	4	2.1	$\kappa = 75$ N
Standing-Twist	177	4	2.1	$\kappa = 125$ N
Jumping	489	4	1.9	$\kappa = 100$ N

Table I. : Tree Statistics. The table provides information on the trees for our controllers. The “Size” field gives the number of nodes in the tree, the “Height” field gives the height of the tree, the “Depth” field gives the average depth of the nodes, the “Subset” field gives the largest subset parameter reached during the DOA expansion process.

Nbr Samples	500	1000	2500	5000	30000
Nbr TOPTs RRTFC	57	32	29	24	19
Nbr TOPTs DFC	22	18	19	19	13
Reduction	61%	44%	35%	21%	32%

Table II. : RRTFC vs DFC. The table compares the efficiency of RRTFC and DFC when performing DOA expansion on the domain $\alpha = 0.5$ for one of our getting up controllers. We give the number of TOPTs necessary to cover the domain, averaged over 3 runs. We use increasingly better trajectory optimizers that are obtained by increasing the total number of samples in the CMA optimization. The “Reduction” field gives the percentage of fewer TOPTs when using DFC over RRTFC.

7. DISCUSSION

RRTs have the property to first grow towards the unexplored regions of the domain and then to fill the regions in between branches. It is in the first stage that DOA expansion with an RRT is particularly inefficient. Once the exploration stage is over, the RRT progressively becomes more dense and, consequently, more effective. For instance, using the tree constructed for one of the getting up controllers, we note that only 1 out of 10 TOPTs succeed when the tree has only 2 nodes, while 7 out of 10 TOs succeed when the tree has 80 nodes. The difference between the DFC and the RRTFC becomes more pronounced with larger domains because they imply a longer exploration stage.

In this work, we use a shooting method for the TOPT and PD control for feedback, but other methods could have been employed. As was pointed out by Tedrake [Tedrake 2009], the core DOA expansion algorithms are compatible with a number of methods, some of which will be better suited for certain applications than others. It would be valuable to compare the efficiency of the expansion process when using different methods, for example, LQR control instead of PD control.

Our synthesized motions do not always look natural. Nonetheless, DOA expansion is a computationally tractable way to bring states inside the DOA when they were previously outside. We now discuss current avenues of research to achieve more natural results. The first is to perform DOA expansion on a set of controllers instead of a single controller. Humans do not have a single way to perform a motion, yet it is the assumption made when performing DOA expansion on a single controller. The second is to use a motion capture database to guide the TOPTs to more human-like solutions. The last is on how to determine movement duration in TOPT, i.e., how long should it take for the character to move from one state to another? In this work, we use a constant movement duration whenever the distance between the states is within a threshold. It would be valuable to develop a method that either efficiently optimizes for a cost of time objective term or estimates the movement duration from prior data.

It is possible to modify the DFC algorithm to perform DOA expansion on states that are more likely to occur in practice by changing the sampling distribution in Sec. 4.3. It may, however, be useful to sample states that are unlikely to occur to then bring other states inside the DOA. One possible approach is to sample states that are unlikely to occur if they are in the direction of further, but more likely states. As was pointed out by Glassman et al. [2010], metrics such as dist (Sec. 4.4) can be very inefficient because that they not take the dynamics and constraints of the system into account. Investigating alternative metrics is a possible topic for future work. It would also be valuable to thoroughly experiment more advanced density estimation techniques and different sampling strategies in Sec. 4.3, e.g., quasi-random, MCMC, etc., as was pointed out in Branicky et al. [2001]. The controllers in this paper are all time-based, which are known to lack robustness. It would be valuable to develop a method that efficiently transforms our time-based controllers into robust state-based controllers (e.g., [Grizzle et al. 2010],[Kolter et al. 2008]).

DOA expansion could play an important role in having simulated characters perform skills from a large number of situations and transition between different skills. While previous work were limited to low-dimensional problems, we have shown how it can be performed efficiently for a high-dimensional simulated character. We hope that our work will stimulate future research on DOA expansion so that it becomes a common tool in control design.

REFERENCES

- ABE, Y., DA SILVA, M., AND POPOVIĆ, J. 2007. Multiobjective control with frictional contacts. In *Symposium on Computer Animation (SCA)*. Eurographics Association, 249–258.
- AL BORNO, M., DE LASA, M., AND HERTZMANN, A. 2013. Trajectory optimization for full-body movements with complex contacts. *Visualization and Computer Graphics, IEEE Transactions on* 19, 8, 1405–1414.
- AL BORNO, M., FIUME, E., HERTZMANN, A., AND DE LASA, M. 2014. Feedback control for rotational movements in feature space. In *Computer Graphics Forum*. Vol. 33. Wiley Online Library, 225–233.
- BRANICKY, M. S., LAVALLE, S. M., OLSON, K., AND YANG, L. 2001. Quasi-randomized path planning. In *International Conference on Robotics and Automation (ICRA)*. Vol. 2. IEEE, 1481–1487.
- COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2010. Generalized biped walking control. *ACM Transactions on Graphics* 29, 4, 130.
- FIRMIN, M. AND VAN DE PANNE, M. 2015. Controller design for multi-skilled bipedal characters. *Computer Graphics Forum (to appear)*.
- GLASSMAN, E. AND TEDRAKE, R. 2010. A quadratic regulator-based heuristic for rapidly exploring state space. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5021–5028.
- GRIZZLE, J. W., CHEVALLEREAU, C., AMES, A. D., AND SINNET, R. W. 2010. 3d bipedal robotic walking: models, feedback control, and open problems. In *IFAC Symposium on Nonlinear Control Systems*. Vol. 2. 8.
- HÄMÄLÄINEN, P., ERIKSSON, S., TANSKANEN, E., KYRKI, V., AND LEHTINEN, J. 2014. Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics* 33, 4, 51.
- HANSEN, N. 2006. The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. 75–102.
- HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *ACM Siggraph Computer Graphics*. ACM, 71–78.
- KOLTER, J. Z., COATES, A., NG, A. Y., GU, Y., AND DUHADWAY, C. 2008. Space-indexed dynamic programming: learning to follow trajectories. In *International Conference on Machine Learning (ICML)*. ACM, 488–495.
- KUFFNER, J. J. 2004. Effective sampling and distance metrics for 3d rigid body path planning. In *International Conference on Robotics and Automation (ICRA)*. Vol. 4. IEEE, 3993–3998.
- LAVALLE, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. In *Technical Report*. Computer Science Department, Iowa State University, 11.
- LAVALLE, S. M. AND KUFFNER, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20, 5, 378–400.
- LIN, W.-C. AND HUANG, Y.-J. 2012. Animating rising up from various lying postures and environments. *The Visual Computer* 28, 4, 413–424.
- LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012a. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Transactions on Graphics* 31, 6, 154.
- LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012b. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Transactions on Graphics* 31, 6, 154.
- MORDATCH, I., TODOROV, E., AND POPOVIĆ, Z. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics* 31, 4, 43.
- TASSA, Y., EREZ, T., AND TODOROV, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4906–4913.
- TEDRAKE, R. 2009. Lqr-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems (RSS)*.
- TODOROV, E., EREZ, T., AND TASSA, Y. 2012. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5026–5033.
- WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2010. Optimizing walking controllers for uncertain inputs and environments. *ACM Transactions on Graphics* 29, 4.
- WITKIN, A. AND KASS, M. 1988. Spacetime constraints. In *ACM Siggraph Computer Graphics*. Vol. 22. ACM, 159–168.
- YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics* 26, 3, 105.