Domain of Attraction Expansion for Physics-Based Character Control

MAZEN AL BORNO
University of Toronto
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 subset of the states of the character, known as the *domain of attraction* (DOA). This article shows how many 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*

General Terms: Human Motion Modeling, Physics-based Character Animation

Additional Key Words and Phrases: Computer animation, physics simulation, character animation

This research was supported by NSERC Discovery RGPIN-2014-1507 and RGPIN-2015-04843.

Authors' addresses: M. Al Borno and E. Fiume, Department of Computer Science, University of Toronto, 40 St-George Street, Toronto, Ontario, Canada, M5S 2E4; emails: {mazen, elf}@dgp.toronto.edu; M. van de Panne, Department of Computer Science, University of British Columbia, 2366 Main Mall, Vancouver, British Columbia, Canada, V6T 1Z4; email: van@cs.ubc.ca.

ACM Reference Format:

Mazen Al Borno, Michiel van de Panne, and Eugene Fiume. 2017. Domain of attraction expansion for physics-based character control. ACM Trans. Graph. 36, 2, Article 17 (March 2017), 11 pages.
DOI: http://dx.doi.org/10.1145/3009907

1. INTRODUCTION

It is difficult to find control solutions for the motions of high-degree-of-freedom humanoid characters, as they form small, elusive manifolds within a highly nonlinear motion domain. What is worse is that they often lack robustness, for example, the motions may fail with the smallest push, and they can be difficult to sequence into composite motions. As such, many potential control solutions are not feasible or practical when applied to human motion synthesis. Given a character with skills, for example, balancing, jumping, rising, and so on, a certain set of initial states can achieve the desired motion that we assume is described in terms of goal 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 invoked from states outside the DOA do not achieve a desirable outcome. This article deals with the following question: How can skills be successfully invoked from a much larger set of initial states? In other words, how can we expand the DOA?

Inspired by Tedrake [2009], we present an algorithm to do this using random trees. One contribution of this article is to show how DOA expansion can be performed for a 25-degrees-of-freedom (underactuated) character, in contrast to previous work on fully actuated systems with fewer degrees of freedom. Second, 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 can broadly be characterized as a combination of random sampling and iterative deepening. The algorithm biases the trees to grow in breadth before depth, while RRTs are biased in the converse direction. The key intuition is that *dense* random trees make it easier to steer states inside the DOA.

The goal of DOA expansion is to cover as large a portion as possible of a given domain of interest, that is, 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, that is, locally optimal. We perform DOA expansion on a number of dynamic motions: getting up, crouch-to-stand, jumping, and standing-twist. Our controllers are obtained by optimizing a time-indexed spline that provides proportional-derivative (PD) target angles to achieve a final target state. The method could equivalently be applied to perform DOA expansion on other types of initial controllers, for example, Coros et al. [2010] and 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 find the appropriate time-indexed target PD-reference trajectory for control.

2. RELATED WORK

Physics-based motion can be synthesized by optimizing a trajectory, for example, Witkin and Kass [1988], Mordatch et al. [2012], and Al Borno et al. [2013] or by designing control laws for specific movements, for example, Yin et al. [2007], Hodgins et al. [1995], and Al Borno et al. [2014]. In both of these categories, data-driven methods have been explored to improve the realism of the synthesized motions, for example, Lee et al. [2010] and Liu et al. [2010]. Open-loop controllers usually have small or negligble DOAs, while closed-loop controllers such as SIMBICON [Yin et al. 2007] can have large DOAs. Corresponding to these categories 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, 2015] are examples of MPC methods. The iterative Linear-Quadratic-Gaussian (iLQG) method of Tassa et al. can have the character get up to a standing position from arbitrary lying positions on the ground. However, the character gets up with a single bounce, implying that 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, for example, ± 300 Nm. The multimodal sampling method of Hämäläinen et al. [2014] can have the character balance and get up from a wide variety of scenarios. In Hämäläinen et al. [2015], multimodal sampling is combined with a Markov Random Field factorization to enable the character to balance on a ball and recover from disturbances in real time. These MPC-based methods fail to have the character get up or balance in some cases. It is for such failure cases that offline pre-computation is necessary. We discuss our results in relation to MPC methods in further detail in Section 7.

Our work falls in the category of control policies that leverage offline pre-computation. Wang et al. [2010] optimize the parameters of single controller given a distribution of initial states, but the method does not generalize well to states that are far away from the motions for which the original controller was designed. Sok et al. [2007] construct a control policy from optimized trajectories that track motion capture data but does not tackle the case of the optimization falling in local minima. Our work shows how trajectories can be connected together to avoid local minima, without the use of prior motion data. Atkeson et al. [2008] 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 [2009] foregos the goal of finding globally optimal policies for the goal of finding good-enough policies instead. The principal 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] present 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, for example, Felis and Mombaur [2016], linear quadratic regulators (LQR) for feedback control around these trajectories, and Lyapunov functions for estimating the DOA. Our work uses the model-free shooting method of Al Borno et al. [2013] to synthesize trajectories, stabilizes motions using PD controllers indexed by time and employs forward dynamics to evaluate the DOA. The method provides an approach that fundamentally differs from the single-optimized-trajectory that results from MPC; it develops a tree of reference trajectories that is computed offline and then exploits this tree online. While previous character animation articles have produced workable DOAs for given scenarios, they do not provide practical mechanisms for significantly growing the DOA. Our work does not currently use motion capture data, but their use is not precluded; they could be used to guide our trajectory optimization towards realistic solutions, for example, Ju et al. [2013].

An important and under-studied problem in character animation and robotics is that of how to sequence controllers over time, that is, how can a character chain skills such as getting up followed by a standing-twist and a jump? Getting up motions are synthesized by Lin et al. [2012], where motion capture data of the character getting up from a given initial state is extended to new lying postures by using physics simulation, producing a final motion that has both physics-based and motion-capture-based segments. In contrast, the motions resulting from the sequencing of the controllers that we develop are entirely physically based, requiring no motion capture stitching. In related work, Liu et al. [2012] and Ha et al. [2014] use policy search to optimize for linear feedback matrices and controller parameters to achieve very specific motion transitions. Firmin et al. [2015] provide a control language for skill authoring, where the resulting DOA depends on the skill of the designer. Faloutsos et al. [2001] solve a binary classification problem with support vector machines to predict if controllers can be sequenced together. These approaches are less generic in several respects than the one we develop in this article, wherein DOA expansion can be used to achieve controller sequencing, among other capabilities.

3. DOMAIN OF ATTRACTION EXPANSION

Our goal is to perform DOA expansion on a given initial controller in order to make it suitable for a larger set of initial states. The main idea in the expansion process is to sample a state outside the DOA, connect it with the current DOA using TOPT, apply feedback control on the trajectory so a nearby region is now also inside the DOA, and repeat this process by sampling again. In this section, we present algorithms to construct random trees offline that cover as much as possible of a domain of interest in state space. In Section 4, we describe the techniques developed to implement the various operations in the algorithms such as how to perform TOPT, obtain a feedback controller, and so on. We also show how the trees can be used in near real-time for character control. In Section 5 and Section 6, we analyze the performance of the algorithms on a low-dimensional canonical sytem (a pendulum) and on a simulated character.

We now begin to outline 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,

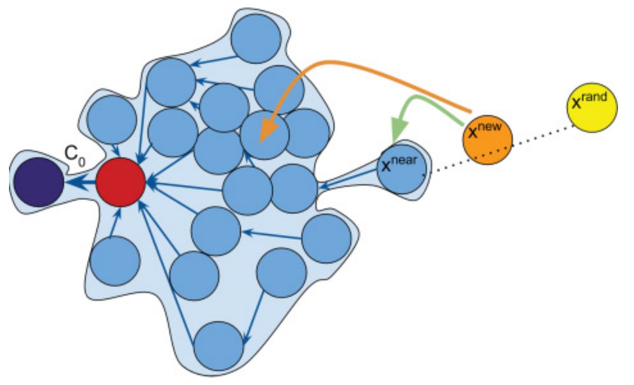

Fig. 1. DOA expansion with an RRT. An RRT starts by sampling a random state in the domain $\mathbf{x}^{rand}$ and finds its closest node $\mathbf{x}^{near}$ in the tree. It then attempts to connect to $\mathbf{x}^{near}$ (green arrow) starting from a state that is close and in the direction of $\mathbf{x}^{rand}$. The intuition behind the Dense Feedback Coverage algorithm 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.

when using the controller from that state, the character's end state is inside a given goal set, that is, within an epsilon distance of a goal state $\mathbf{x}^{goal}$. We specify a single goal state for our controllers, but it is possible to handle multiple goal states simply by checking if an end state is within an epsilon distance of any one of a given list of goal states. Let T_0 denote the duration required to bring $\mathbf{x}^{target}$ to the goal set using $\mathbf{C}_0$.

ALGORITHM 1: RRTFC Algorithm

```
1:  $\Gamma$.add_node($\mathbf{x}^{goal}$, NULL, NULL, 0)
2:  $p \leftarrow$ pointer to the root
3:  $\Gamma$.add_node($\mathbf{x}^{target}$, $\mathbf{C}_0$, $p$, $T_0$)
4:  for $k = 0, \ldots, K$ 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:         $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T \leftarrow$ 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 $\mathbf{x}^{rand}$ inside Ω, finding its closest state in the tree and obtaining state $\mathbf{x}^{new}$ by interpolation, as illustrated in Figure 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.

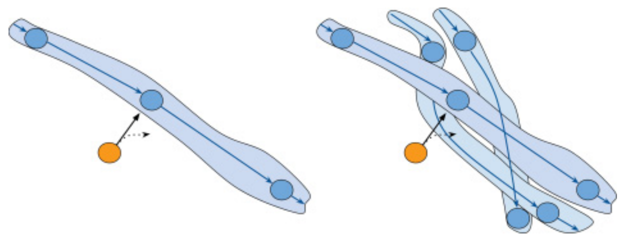

Fig. 2. Examples of trajectory optimization in a sparse and dense tree. The blue tubes illustrate the domain covered by edges of the tree. The orange circle is a random state that we attempt to connect to its nearest state in the tree with trajectory optimization. The dotted curves are the trajectories 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).

A 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 Figure 2). We do not test if the states at all previous timesteps are inside the DOA because it would be too expensive. If the connection is successful, then the states at all timesteps are added to the tree, even if they are not inside Ω. This increases the likelihood of $\mathbf{x}_T$ of a future attempt to land inside the DOA.

3.2 Dense Feedback Coverage Algorithm

RRTs have the property of growing long branches quickly to efficiently explore the domain and to find a path to a given state. This property is less relevant in the context of DOA expansion, where the objective is to find a path to every state in the domain as opposed to a single 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 Figure 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 $\Omega_0 \subset \Omega_1 \subset \ldots \subset \Omega$, where Ω_0 denotes the DOA of the initial controller (see Figure 3 (left) and Section 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 in a region 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 sees a larger target area to perform new connections. This is similar to the growth of the solution region in dynamic programming methods. To solve the problem of bringing a state in a given domain inside the DOA, we use the solutions of subproblems found for smaller subsets of the domain that lie closer to the goal state.

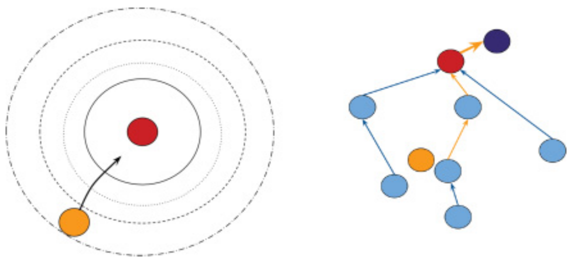

Fig. 3. (Left) DOA coverage with DFC. The DFC algorithm progressively covers the domain around the target state (the red circle). Increasingly larger subsets are illustrated in the figure. The orange circle is a random state sampled on the largest subset. TOPT is used to steer the state to its closest state in a smaller subset. (Right) Determining if a state is inside the DOA. To determine whether a state $\mathbf{x}^r$ (the orange circle) is inside the DOA, we perform simulation rollouts, starting from the controllers associated with the closest states in the tree (the orange arrows illustrate the sequence of controllers used for one closest state). We use PD controllers with targets provided by time-indexed splines. A rollout consists of moving up the nodes of a branch, until the root (the goal state) is reached.

ALGORITHM 2: Dense Feedback Coverage Algorithm

```
 1: $\Gamma$.add_node($\mathbf{x}^{goal}$, NULL, NULL, 0)
 2: $p \leftarrow$ pointer to the root
 3: $\Gamma$.add_node($\mathbf{x}^{target}$, $\mathbf{C}_0$, $p$, $T_0$)
 4: Divide the domain in sets $\Omega_0 \subset \Omega_1 \subset \Omega_2 \subset \ldots \subset \Omega_N$, where
    $\Omega_N = \Omega$
 5: for $i = 1, \ldots, N$ do
 6:     for $k = 0, \ldots, K_i$ 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:             $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T \leftarrow$ full trajectory from the TOPT
12:             if $\mathbf{x}_T \in \chi$ then
13:                 Obtain a feedback controller $\mathbf{C}$
14:                 $p \leftarrow$ pointer to the node that has $\mathbf{x}_T$ in its DOA
15:                 $\Gamma$.add_node($\mathbf{x}_1$, $\mathbf{C}$, $p$, $T$)
16:                 $\Gamma$.add_node($\mathbf{x}_2$, $\mathbf{C}$, $p$, $T-1$)
17:                 ...
18:                 $\Gamma$.add_node($\mathbf{x}_{T-1}$, $\mathbf{C}$, $p$, 1)
    return $\Gamma$
```

The optimal number of subsets Ω_k will depend on the problem. Generally, choosing more subsets yields more samples that are already inside the DOA, while choosing to have fewer 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, that is, too far away from $\mathbf{x}^{rand}$, will increase the likelihood of a failed connection. One strategy is to choose the smallest subset such that the distance between $\mathbf{x}^{rand}$ and $\mathbf{x}^{near}$ is within a specified value. We typically simply choose the nearest neighbor to be inside subset Ω_{i-1} when expanding subset Ω_i.

If the DOA of the initial controller is particularly small, then it may 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, for example, by using RRTFC on Ω_1. We then continue the DOA expansion process on the entire domain with DFC. This strategy is used for the pendulum. For the simulated character, we directly employ DFC on the entire domain.

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 $\Omega_i \backslash \Omega_{i-1}$, that is, 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}, then we can have 5% of the samples in Ω_{i-1} and 95% in Ω_i. Section 4.8 describes how we can estimate the domain coverage.

4. TECHNIQUES

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

4.1 Trajectory Optimization and Feedback Control

We build on the method of Al Borno et al. [2013], which optimizes a reference trajectory $\hat{\mathbf{q}}_{1:T}$ represented by a cubic B-spline. The output motion $\mathbf{x}_{1:T} = (\mathbf{q}_{1:T}, \dot{\mathbf{q}}_{1:T})$ is computed by forward dynamics. The torque u_t for a joint is determined by PD control:

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

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

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

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

Note that this method returns both the trajectory and the closed-loop controller (Equation (1)) in the RRTFC (lines 9 and 12) and DFC (lines 10 and 13) algorithms. Although our joint-local PD controllers are weak, that is, 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, for example, as in Liu et al. [2012]. In general, there exists a tradeoff between learning a small number of robust controllers or to instead develop a control policy using many weak controllers.

A simple extension to the method of Al Borno et al. [2013] can also allow for the optimization of the motion duration. We add a time variable to the optimization and a time-cost objective that penalizes excessively slow movements. Specifically, we add a cost term $w_3 T$ to Equation (2). However, 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 then a chosen threshold, we use a constant movement duration, for example, 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 an explicit model as used in previous work [Tedrake 2009] and that does not currently scale to high dimensions. We begin 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 Section 4.1. The rollouts proceed until the root of the tree is reached (see Figure 3 (right)). 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 given in Algorithm 3. We consider the V closest states instead of the closest state alone because the distance metric is in general imperfect. We often find $\mathbf{x}^r$ to be inside the DOA of a different node than the "closest" node. We typically choose $V \in [50, 250]$. With a larger value, it is more likely to find a path that leads to the goal set, but the DOA expansion process is also more time-consuming.

4.2.1 *Minimal Time.* Note that Algorithm 3 halts with the first rollout that 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 a time objective. While this slows the DOA expansion process, it 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 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 in real time on average 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, for example, 4s. We have tried learning a nearest-neighbor classifier that maps a state to a controller, that is, 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 ← false
2: Parent ← 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 $n \neq$ 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 dist($\mathbf{x}^f, \mathbf{x}^{goal}$) $< \epsilon$ then
12:        IsInside ← true
13:        Parent ← pointer to the $i$th closest node
14:        break
```

ALGORITHM 4: [IsInside, Parent] = FindBestStartState($\mathbf{x}^r$)

```
1: IsInside ← false
2: Parent ← 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 $n \neq$ 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 dist($\mathbf{x}^f, \mathbf{x}^{goal}$) $< \epsilon$ then
15:        IsInside ← true
16:        if $t < s$ then
17:            $s \leftarrow t$
18:            Parent ← 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 parameterize 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 parameterizes Ω_i, m_j is the mass of the associated body link, and l_j and u_j are lower and upper joint limits. For instance, $\alpha = 5$ creates angle ranges of 27° for the hips, 46° for the knees, and 90° for the shoulders, subject to 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 parameterized 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 for a short time duration. The state of the character lying on the ground is our random sample. For the other controllers (crouch-to-stand, jumping, etc.), the states are sampled as follows to increase the robustness to external disturbances. 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 taken as the state occurring 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 parameterizes Ω_i. The values of α_i and κ_i increase with i (see Figure 8).

4.4 State Distance Metric

Given a random state $\mathbf{x}^{rand}$, the DOA expansion algorithms need to find the 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,

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

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 possible heuristic for yielding improved motions is to choose N nearest neighbors instead of one, and solve for N TOPTs. Of the successful connections, we then select 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 achieved for $N = 1$ and $N = 10$.

4.6 Connecting Controllers

An important application of DOA expansion is its ability to sequence different controllers. If the terminal state of a controller is close to the 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 Section 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 thus far requires pre-specifying a controller 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, that is, combine MPC methods with *some* offline pre-computation to avoid both the curse of dimensionality and the local minima. The core idea is to do offline construction of a small tree that sketches the general path to arrive at the goal state. Online optimization is then used to track a branch in the tree. This hybrid approach is slower than the pure offline optimization approach because it requires some online optimization.

To determine if a state $\mathbf{x}^r$ is inside the DOA in the hybrid of offline and online optimization (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 $\mathbf{x}^{goal}$, instead of performing rollouts with the controllers of the closest nodes (line 10 in Algorithm 4). For the simulated character, we perform 100 CMA iterations of 30 short rollouts (e.g., 0.7s) in the method described in Section 4.1, which runs at about 5% real time on a machine with eight logical cores. Fewer iterations would be required for longer rollouts or different methods could be used, for example, Hämäläinen et al. [2014]. The cost function used is Equation (2), with $\mathbf{x}^d = \mathbf{x}^{goal}$. At each timestep t, the control torque in the rollouts is determined by $\mathbf{u}_t = \mathbf{u}_t^{ng} + \mathbf{u}_t^*$, where $\mathbf{u}_t^{ng}$ is the control torque determined from the PD controllers associated with the branch from $\mathbf{x}^{near}$ to $\mathbf{x}^{goal}$, and $\mathbf{u}_t^*$ is the newly optimized portion of the control torque (i.e., determined from Equation (1) with the new reference trajectory). We found that this technique returns a better solution (measured in terms of the cost function) than to simply initialize the new trajectory optimization with its nearest neighbor's solution. In one experiment, this approach returns a better solution 9 times of 10 when $\mathbf{x}^r$ is sampled near $\mathbf{x}^{near}$ using the approach described in Section 4.3 with $\kappa = 0.5$. When $\mathbf{x}^r$ is sampled further away from $\mathbf{x}^{near}$ (e.g., by using $\kappa = 2.5$), both approaches have comparable performances.

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 by using the tree online (Section 4.2.2). The proportion of samples inside the domain provides the estimate of domain coverage.

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 3.10kg and the torque limits are $\pm$200Nm. The goal is to stabilize the pendulum to its unstable equilibrium $[\pi, 0]$, that is, 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 Section 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 into four 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 $k = 1, \ldots, 4$. 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, that is, 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 into 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 3.10kg. 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 Figure 6). Our initial controllers are synthesized by manually specifying between one and three key-frames as targets in a TOPT problem (Section 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

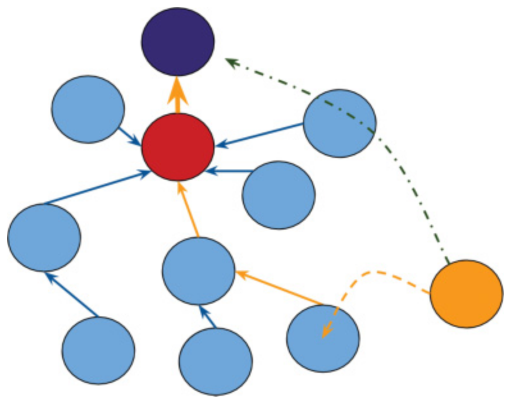

Fig. 4. Full branch rollouts. In this figure, we attempt to connect the orange node to its nearest node in the tree. The typical approach when performing a TOPT is to steer the node as close as possible to the nearest node (the dotted orange arrow) to enter its DOA. In some problems (crouch-to-stand, jumping, standing-twist), we were not able to make any successful connection this way. We modify the TOPT by including the controllers associated with the closest branch in the rollouts of the TOPT (orange arrows). The objective of the TOPT is to steer the character as close as possible to the goal state. This optimization is less expensive and more likely to succeed than searching for a direct path to the goal state (green arrow).

the controllers can be used. We also regulate the final global orientation of the character, that is, it must get up, twist, and so on, with 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 more challenging than the getting up controllers, likely because momentum effects are more present. The initial controllers have very small DOAs, making it 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 $\mathbf{x}_1$. For the modified case, our TOPTs will minimize the distance to $\mathbf{x}^{goal}$, using $\mathbf{C}$ after a short time interval of 0.2s elapses in the rollouts. In other words, the TOPT optimizes the torques used in the preliminary time interval; the following torques are specified by $\mathbf{C}$. This significantly slows down the TOPT since we are now performing rollouts for the entire duration of the branch of the tree instead of the edge only (see Figure 4). Once the DOA of the tree grows 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 until the motion comes largely to rest. The target pose provides a convenient repeatable starting state from which to then apply a fixed, known rising strategy, thus avoiding the need for DOA expansion. In the accompanying video, we illustrate that this heuristic method does

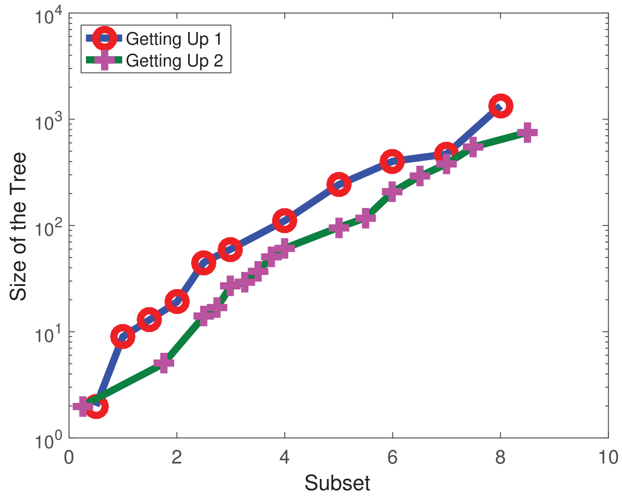

Fig. 5. The size of the tree. The figure plots the size of the tree in log scale required to cover a subset for the "Getting Up 1" and the "Getting Up 2" controllers. The subsets are parametrized by α, which is defined in Section 4.3. In the plot, the value of α is shown in the subset axis.

not work with our controllers. The character never exactly reaches the designated target pose and the resulting errors typically make the controllers fail. Our experiments show that a small tree, for example, 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 0.01s timestep, a coefficient of friction of 1, and torque limits of ±150Nm. Our simulated character has a mass of 40.5kg, a height of 1.61m, and 25 degrees-of-freedom.

7. RESULTS

DOA expansion on the getting up, crouch-to-stand, and standing-twist controllers were performed offline. The offline-online approach (Section 4.7) was used for the jumping controller. A standing controller with a similar approach to Hämäläinen et al. [2014] is used when the character is near the last standing pose. 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, then the method does not attempt to discover how to get up from scratch. Instead, it attempts to find a path to a state where a getting up strategy is available. 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 Figure 6). 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 7.5Ns, 10Ns, and 12.5Ns are applied in random directions on the character (see Figure 7). The results are compared to controllers without DOA expansion. 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

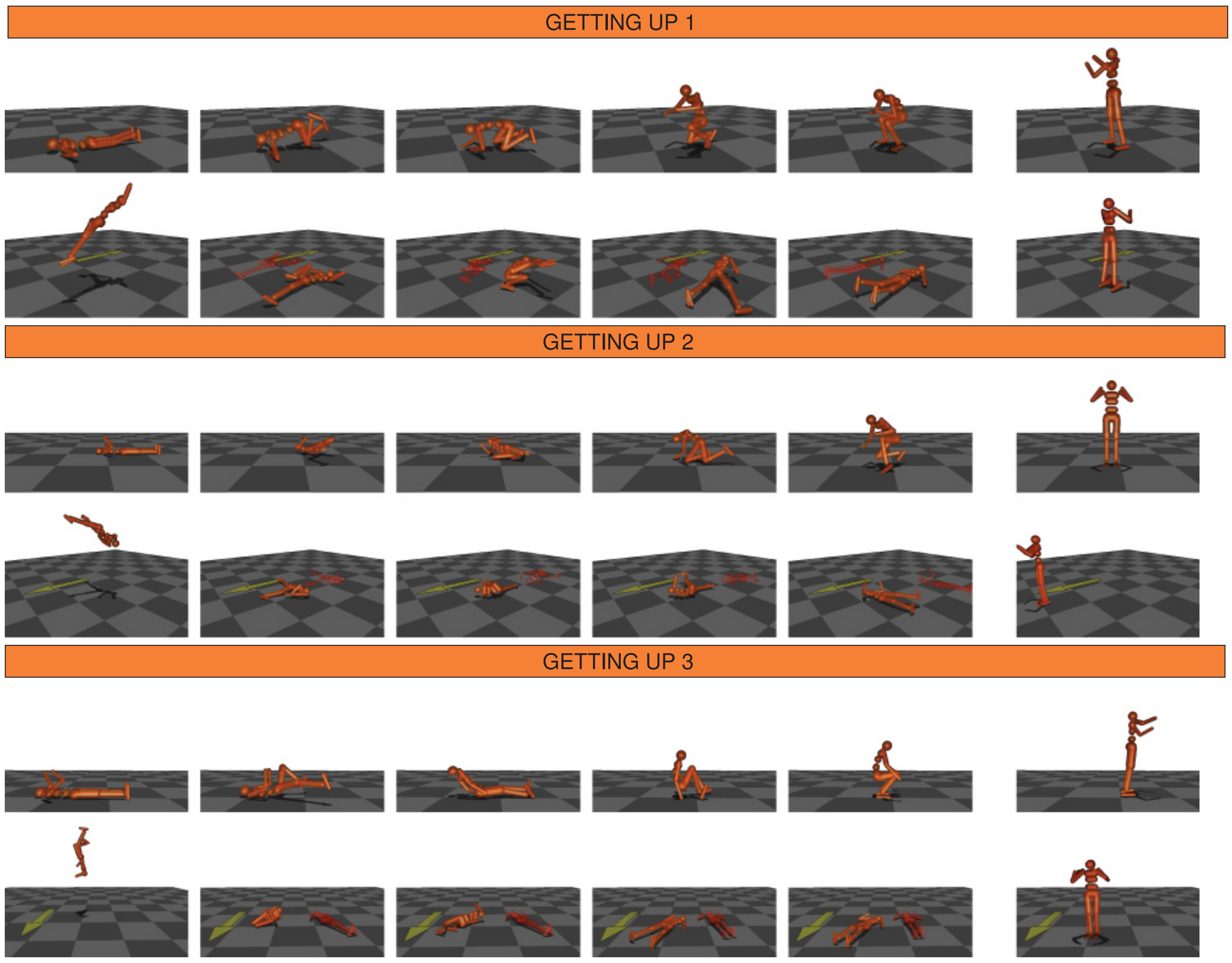

Fig. 6. Getting up controllers. The figure is divided into three pairs of sequences. For each pair, the top sequence illustrates an initial getting up controller, and the bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). 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.

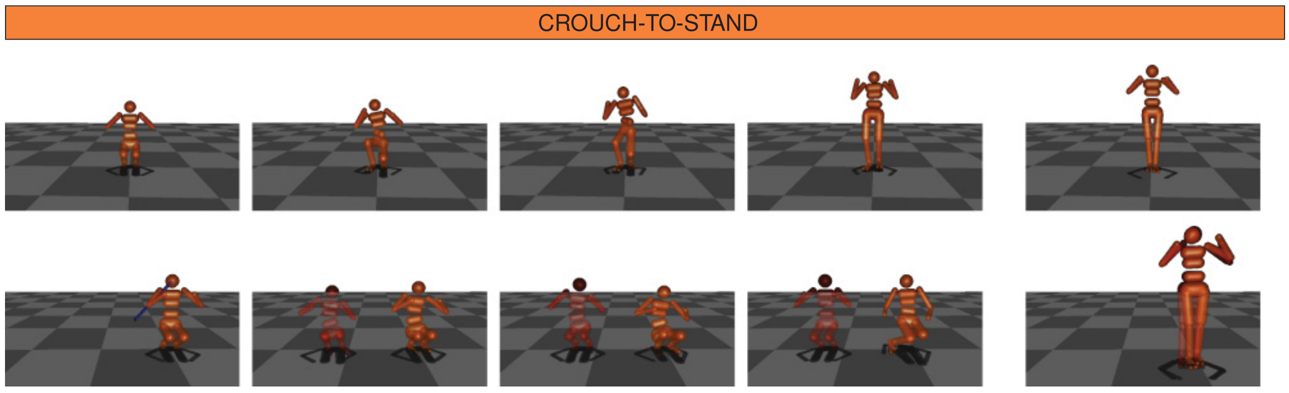

Fig. 7. Crouch-to-stand controller. The top sequence consists of key-frames of the crouch-to-stand controller. The bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The blue line on the first frame is an external force applied on the head. The red character is the pose of the currently active node.

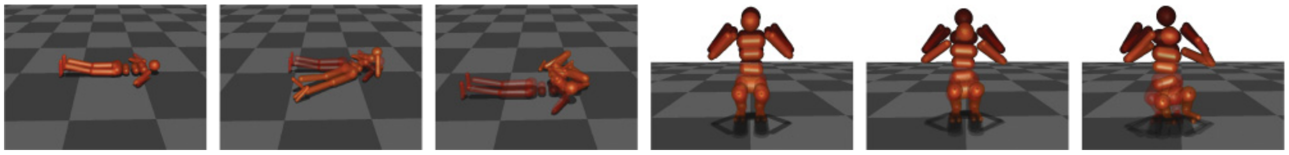

Fig. 8. Exploration regions. The sequence shows typical states (orange character) in progressively larger subsets for the target states (the red character) associated with the "Getting Up 1" and the crouch-to-stand controllers. The subsets are $\alpha = 1$, $\alpha = 5$, and $\alpha = 10$ for the first three figures and $\kappa = 100$Nm, $\kappa = 400$Nm, and $\kappa = 1000$Nm for the last three figures.

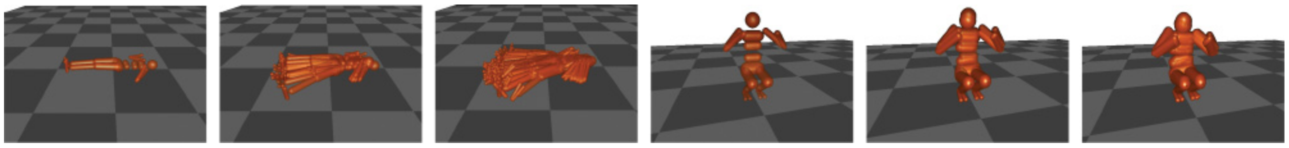

Fig. 9. Visualizing the nodes in the Tree. The first three figures shows the DOA Tree with 1, 10, and 40 nodes for the "Getting Up 2" controller. The last three figures shows the DOA Tree with 1, 10, and 40 nodes for the crouch-to-stand controller. These nodes are sampled from Ω. The difference in velocities between the nodes cannot be visualized.

Table I. Tree Statistics

Controller	Size	Height	Depth	Subset
Getting Up 1	1,377	14	7.3	$\alpha = 8$
Getting Up 2	751	12	5.4	$\alpha = 8.5$
Getting Up 3	277	41	19.3	$\alpha = 10$
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

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, and the "Subset" field gives the largest subset parameter reached during the DOA expansion process.

Table II. RRTFC vs DFC

Number of Samples	500	1,000	2,500	5,000	30,000
# TOPTs RRTFC	57	32	29	24	19
# TOPTs DFC	22	18	19	19	13
Reduction	61%	44%	35%	21%	32%

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 three runs. We use increasingly better trajectory optimizers that are obtained by increasing the total number of samples in CMA. The "Reduction" field gives the percentage of fewer TOPTs when using DFC over RRTFC.

standing-twist transition controller. We also show how jumping and stand-crouch controllers can be used repeatedly in a sequence.

The number of nodes required to cover a given subset depends on the controller. This can be seen in Figure 5, where we plot the total number of nodes required to cover progressively larger subsets for some of our controllers. Intuitively, we would expect the rate of growth of the tree to increase with larger subsets. The plot suggests that the trees grow exponentially with the subset parameter (the vertical axis is in log scale). In Figure 9, we plot some of the nodes in the trees for the "Getting Up 3" and the crouch-to-stand controllers to help visualize the expansion process. Table I provides some statistics on the trees constructed for our controllers. The trees are constructed by increasing α and κ by increments of 0.25–1 and 25Nm, respectively. The values are chosen so the distance between two states in subsequent subsets would not be too difficult for the TOPT. The trees required between 1 and 3 days of computation on a desktop computer.

8. COMPARISONS

The discovered solutions with DOA expansion include the character taking a step or leaning on its knee to maintain 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. More extensive searches allow offline methods to avoid some of these local minima. In one experiment, we use the SMC method to steer randomly sampled initial states to the DOA of a getting up controller. The method succeeds 9, 4, and 2 times of 10 trials when the sampled states are inside the subsets $\alpha = 2, 6, 8$, respectively. After performing DOA expansion, we succeed 10, 10, and 9 times for the same subsets. The failure case in the DOA method is due to not performing enough iterations offline or failing to find the node that leads the state inside the DOA online (Section 4.2.2). 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 actuation approach, which is specific to the Open Dynamics Engine physics simulator. DOA expansion can be used to steer the "failure" states inside the DOA of these MPC methods.

The quality of the trajectory optimizer, that is, how closely it can 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 five CMA optimizers of increasing quality that are obtained by increasing the total number of samples in the rollouts. The optimizers are then used to perform DOA expansion on the domain $\alpha = 0.5$ for one of our getting up controllers (Getting Up 2). Table II provides the number of TOPTs ($n_s + n_f$ in Section 5) required to cover the domain as averaged over three runs. The duration of the rollouts is set to 0.2s so 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. We also compared RRTFC and DFC when covering a larger domain, namely $\alpha = 10$, using the CMA optimizer with 500 samples. The DFC method required 562 TOPTs, as opposed to 4,763 for RRTFC, which constitutes an 88% reduction in the computational cost of DOA expansion. For this problem, we found that only 5 of 40 successful TOPTs actually connect the random sample $\mathbf{x}^{new}$ to the target node $\mathbf{x}^{near}$ (see Section 3.1); the rest connect to other nodes in the tree.

9. 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 of 10 TOPTs succeed when the tree has only 2 nodes, while 7 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 pointed out by 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 use a motion capture database to guide the TOPTs to more human-like solutions. The second 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 last avenue is to better determine movement duration in TOPT, that is, 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 we describe in Section 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 (Section 4.4) can be very inefficient, because they do not take the dynamics and constraints of the system into account. Investigating alternative metrics remains a topic for future work. It would also be valuable to thoroughly experiment more advanced density estimation techniques and different sampling strategies in Section 4.3, for example, quasi-random or Markov chain Monte Carlo, as was pointed out in Branicky et al. [2001]. The controllers in this article are all time indexed, which are known to lack robustness. It would be valuable to develop a method that efficiently transforms our time-indexed controllers into robust state-based controllers (e.g., Grizzle et al. [2010] and 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 in this area was 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 it becomes a common tool in control design.

ACKNOWLEDGMENTS

We are grateful to the reviewers and the associate editor for their comments and suggestions to improve the manuscript.

REFERENCES

Yeuhi Abe, Marco da Silva, and Jovan Popović. 2007. Multiobjective control with frictional contacts. In *Symposium on Computer Animation (SCA)*. Eurographics Association, 249–258.

Mazen Al Borno, Martin De Lasa, and Aaron Hertzmann. 2013. Trajectory optimization for full-body movements with complex contacts. *IEEE TRans. Vis. Comput. Graph.* 19, 8 (2013), 1405–1414.

Mazen Al Borno, Eugene Fiume, A. Hertzmann, and M. de Lasa. 2014. Feedback control for rotational movements in feature space. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 225–233.

Chris Atkeson and Benjamin Stephens. 2008. Random sampling of states in dynamic programming. In *Advances in Neural Information Processing Systems*. 33–40.

Michael S. Branicky, Steven M. LaValle, Kari Olson, and Libo Yang. 2001. Quasi-randomized path planning. In *International Conference on Robotics and Automation (ICRA)*, Vol. 2. IEEE, 1481–1487.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized biped walking control. *ACM Trans. Graph.* 29, 4 (2010), 130.

Petros Faloutsos, Michiel Van de Panne, and Demetri Terzopoulos. 2001. Composable controllers for physics-based character animation. *ACM Trans. Graph.* (2001), 251–260.

Martin Felis and Katja Mombaur. 2016. Synthesis of full-body 3-D human gait using optimal control methods. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1560–1566.

Michael Firmin and Michiel van de Panne. 2015. Controller design for multi-skilled bipedal characters. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 50–63.

Elena Glassman and Russ Tedrake. 2010. A quadratic regulator-based heuristic for rapidly exploring state space. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5021–5028.

Jessy W. Grizzle, Christine Chevallereau, Aaron D. Ames, and Ryan W Sinnet. 2010. 3D bipedal robotic walking: models, feedback control, and open problems. In *IFAC Symposium on Nonlinear Control Systems*, Vol. 2. 8.

Sehoon Ha and C. Karen Liu. 2014. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Trans. Graph.* 34, 1 (2014), 1.

Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. 2014. Online motion synthesis using sequential Monte Carlo. *ACM Trans. Graph.* 33, 4 (2014), 51.

Perttu Hämäläinen, Joose Rajamäki, and C. Karen Liu. 2015. Online control of simulated humanoids using particle belief propagation. *ACM Trans. Graph.* 34, 4 (2015), 81.

N. Hansen. 2006. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. 75–102.

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. 1995. Animating human athletics. In *ACM Siggraph Computer Graphics*. ACM, 71–78.

Eunjung Ju, Jungdam Won, Jehee Lee, Byungkuk Choi, Junyong Noh, and Min Gyu Choi. 2013. Data-driven control of flapping flight. *ACM Trans. Graph.* 32, 5 (2013), 151.

J. Zico Kolter, Adam Coates, Andrew Y. Ng, Yi Gu, and Charles DuHadway. 2008. Space-indexed dynamic programming: Learning to follow trajectories. In *International Conference on Machine Learning (ICML)*. ACM, 488–495.

James J. Kuffner. 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.

Steven M. LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. In *Technical Report*. Computer Science Department, Iowa State University, 11.

Steven M. LaValle and James J. Kuffner. 2001. Randomized kinodynamic planning. *Int. J. Robot. Res.* 20, 5 (2001), 378–400.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven biped control. In *ACM Trans. Graph.*, Vol. 29. ACM, 129.

Wen-Chieh Lin and Yi-Jheng Huang. 2012. Animating rising up from various lying postures and environments. *Vis. Comput.* 28, 4 (2012), 413–424.

Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. 2012. Terrain runner: Control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph.* 31, 6 (2012), 154.

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based contact-rich motion control. *ACM Trans. Graph.* 29, 4 (2010), 128.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.* 31, 4 (2012), 43.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. In *ACM Trans. Graph.*, Vol. 26. ACM, 107.

Yuval Tassa, Tom Erez, and Emanuel Todorov. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4906–4913.

Russ Tedrake. 2009. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems (RSS)*.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5026–5033.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2010. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.* 29, 4 (2010).

Andrew Witkin and Michael Kass. 1988. Spacetime constraints. In *ACM Siggraph Computer Graphics*, Vol. 22. ACM, 159–168.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.* 26, 3 (2007), 105.

Received November 2015; revised November 2016; accepted December 2016