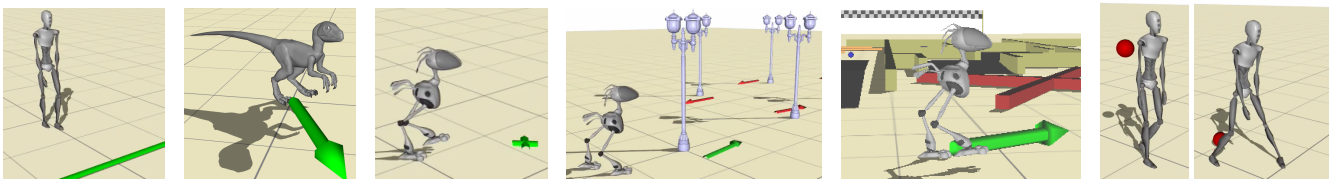
# Robust Task-based Control Policies for Physics-based Characters

Stelian Coros Philippe Beaudoin Michiel van de Panne*

University of British Columbia



(a) Go-to-line (b) Heading (c) Go-to-point (d) Point-with-heading (e) Heading-and-speed (f) Very Robust Walk

**Figure 1:** *We precompute task-specific control policies for real-time physics-based characters. The character moves efficiently towards the current goal, responds interactively to changes of the goal, and can respond to significant physical interaction with the environment.*

## Abstract

We present a method for precomputing robust task-based control policies for physically simulated characters. This allows for characters that can demonstrate skill and purpose in completing a given task, such as walking to a target location, while physically interacting with the environment in significant ways. As input, the method assumes an abstract action vocabulary consisting of balance-aware, step-based controllers. A novel constrained state exploration phase is first used to define a character dynamics model as well as a finite volume of character states over which the control policy will be defined. An optimized control policy is then computed using reinforcement learning. The final policy spans the cross-product of the character state and task state, and is more robust than the conrollers it is constructed from. We demonstrate real-time results for six locomotion-based tasks and on three highly-varied bipedal characters. We further provide a game-scenario demonstration.

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

**Keywords:** Simulation of Skilled Movement, Animation

## 1 Introduction

Human and animal motions are the product of physics and muscular control. However, modeling the control needed to achieve physical simulations of natural, agile motions remains difficult. It is not only necessary to model individual skills such as walking and running, but also to find good ways of integrating these skills in a seamless fashion in order to produce purposeful motion that achieves a given goal or task.

This paper demonstrates the integration of physics-based locomotion skills towards solving tasks such as efficiently moving to a target location or target heading. As they go about their task, the characters can be physically perturbed and have other dynamic interactions with their environment. The individual skills or actions consist of step-based controllers of the type proposed in [Yin et al. 2007] and are assumed to be given as input. Their task-based integration is non-trivial because the individual actions do not directly specify the final motion, as with kinematic models. Instead, they control it indirectly by steering the evolution of the high-dimensional character state. Given the many possible character states and the indirect steering nature of the available actions, this leads to an expansive space of possible motions that is difficult to model. To this end, we describe a process for creating a compact, restricted model of the dynamics with the help of a set of *trusted states*. The dynamics model is then developed by beginning at the trusted states and computing the state closure under the set of available individual actions, subject to a further limited-distance constraint with respect to the trusted states.

The task goal is specified using a reward or cost function, such as the distance to a goal point. The optimized control policy is then computed using fitted value iteration, which is a model-based reinforcement learning algorithm. Importantly, the control policy is defined over the cross product of the character state and the task state. These are both continuously-valued in our case and results in a high dimensional domain for the control policy. We show the feasibility of modeling control policies and value functions in such a case. The control policies also naturally integrate the need to maintain balance with the task objectives. For example, if a character is in a falling state and only one action can be taken to avoid a fall, this is captured by a control policy which is then invariant with respect to the task state for that particular character state.

Our computed control policies add robustness in a number of important ways. First, they exploit actions (controllers) where they are safe to use and avoid their use from states where this would lead to failure. This allows for significant flexibility when designing the individual controllers because there is no need to explicitly supply a model of the preconditions for individual controllers. Second, al-

*e-mail: {scoros|beaudoin|van@cs.ubc.ca}

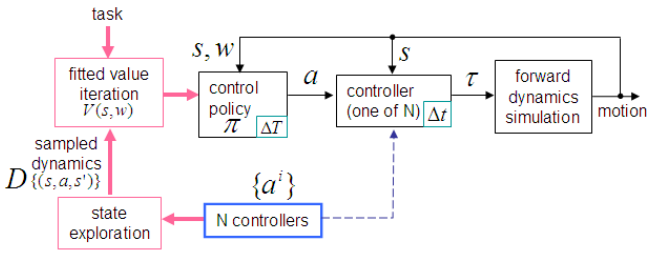



**Figure 2:** *System overview. State exploration and fitted value iteration are performed offline, while the remaining blocks form the real-time control loop. The control policy makes decisions at every character step, $\Delta T$, while the controller makes decisions at every simulation time step, $\Delta t$. $w$ denotes the task state.*

though the form of controllers we use are already quite robust to external perturbations, we show that a task control policy makes them even more robust. Third, our results show that it is now possible to physically perturb simulated bipeds in significant ways while they continually and purposefully attempt to achieve their locomotion-based tasks. We are not aware of other demonstrations of this type of capability across multiple tasks and for multiple characters.

Figure 1 shows examples of tasks and characters for which we compute optimized control policies. The *go-to-line* task consists of walking (or running) forwards or backwards to the goal line. The goal of the *go-to-heading* task is to walk in a specified direction. The *go-to-point* task uses forward, backwards, or sideways walking to move to a goal location anywhere on the plane, and the *go-to-point-with-heading* task has the additional requirement that the character be facing in a given orientation when arriving at the goal. The *heading-and-speed* task walks in a specified direction at a specified speed. For all these tasks, the simulated character can respond in real time to changes of the goal and to physical interactions with the environment, such as stumbling over an object or responding to a push.

### 1.1 Overview

Figure 2 shows a block diagram of the system. The creation of task-based control policies begins with the design of the $N$ *controllers* that will comprise the abstract action vocabulary available to the control policy. The task *control policy*, $\pi(s, w)$, then guides the motion on a step-by-step basis in a way that best achieves the task. Here, $s$ is the character state and $w$ is the task state, a simple example of which is the $(x, y)$ position of the goal relative to the character, e.g., Figure 1(c).

The control policy for a task is pre-computed offline in two stages. First, a *state exploration* stage samples the dynamics of the character in a process that captures the evolution of the character state as different actions are applied. It also models the volume of state-space over which the control policy is to be defined. Given the sampled dynamics and a task description, the second stage computes an optimized control policy. This is accomplished by instancing the sampled dynamics into the task space and computing optimized actions for the cross-product of character states and task states. The control policy is built using *fitted value iteration*, which repeatedly improves upon a value function approximation (§4.3).

## 2 Previous Work

Control strategies have been developed for many physically-simulated motions, including hopping, running, vaulting, and bicycling [Raibert and Hodgins 1991; Hodgins et al. 1995], standing balance [Khatib et al. 2004; Abe et al. 2007], falling and standing up after a fall [Zordan et al. 2005; Faloutsos et al. 2001], and walking [Laszlo et al. 1996; Sok et al. 2007; Yin et al. 2007; da Silva et al. 2008; Coros et al. 2008; Muico et al. 2009]. Many of these control strategies can perform a variety of motions, such as walks of differing styles, and can demonstrate limited transitions between different controllers. Decisions regarding when to enact transitions between different controllers are generally left to the user or a supervisory controller with limited capabilities. Recent work has examined the composition of controllers using value-function-based interpolation [da Silva et al. 2009], instead of viewing it as a sequencing problem.

A number of methods have recently been proposed for task-based control policies using kinematic motion models. These specify the best motion clip to transition to, based on knowledge of the task at hand and the identity of the currently-playing clip [Choi et al. 2003; Lee and Lee 2004; Lau and Kuffner 2005; Ikemoto et al. 2005; Lau and Kuffner 2006; Treuille et al. 2007; McCann and Pollard 2007; Lo and Zwicker 2008; Zhao and Safonova 2008]. Objective functions for this kind of motion planning strike a compromise between the quality of transitions between motion clips (visible jumps are undesireable) and task-related criteria, such as the time or effort used to meet a desired goal. The methods are commonly applied to a graph-based model of possible motions that is instanced in the task space at chosen sample points. The control policy is computed in unison with a value function, which is a function of both the character state, $s$, as represented by the current motion clip, as well as the task state, $w$. Given a finite number of motion clips, $s$ is discrete in nature for most kinematic motion models. However, $s$ is continuous when working with dynamic characters because the result of physical simulations is not fully constrained. As we shall detail later (§6), our controller-based actions behave differently from kinematic motion clips in a number of important respects. Most fundamentally, kinematic task control policies are limited in that they do not allow physical interaction with the surrounding world.

Reinforcement learning has been applied in robotics in order to develop control policies for walking [Atkeson and Stephens 2007; Morimoto and Atkeson 2007; Morimoto et al. 2007; Byl and Tedrake 2008]. Much of this work focuses on developing optimal controllers for steady-state walking, while the primary focus of our work lies with higher-level tasks. Like much of this work, however, our method plans on a step-by-step basis, i.e., using Poincaré sections sampled at foot contact.

Several methods have been developed to enable real and simulated humanoid robots to do online planning for reaching goal locations while also avoiding known static and dynamic obstacles [Chestnutt 2007; Yoshida et al. 2005]. Terrain navigation can be achieved by generating footstep plans. These are then given as input to walking control strategies which may be based on preview-control methods [Kajita et al. 2003] or proprietary methods, i.e., Honda ASIMO. The footstep plans can be generated using A* search [Chestnutt et al. 2005]. or with the help of precomputation [Lau and Kuffner 2006]. However, walking motions for current biped humanoid robots remain quite fragile and demonstrations of robustness to moderate pushes are a recent development. One of our goals is to develop physics-based characters that can cope with significant physical interaction with the environment during the motion, this being one of the primary benefits of using physics-based simulation in applications such as games. The allowable footstep placements for footstep-based planning generally need to be conservatively designed, which helps limit the extent to which the robot state needs to be considered when planning the next step. We allow for controllers and controller transitions that can be quite dynamic but that can fail as a result. Coping with this is then the job of the task-based

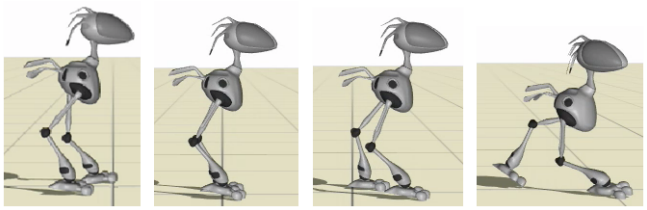


**Figure 3:** *The outcome of an action is highly state dependent in our framework. This example shows the results of the same action applied to four different initial states.*

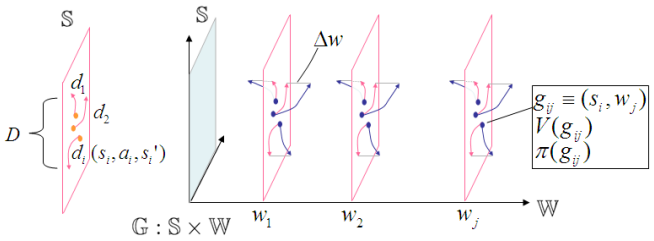



**Figure 4:** *Schematic illustration of how sampled state-to-state dynamics are instanced in the task space. Left: Example transitions as seen in state-space, $\mathbb{S}$. Right: Example transitions as seen in $\mathbb{S} \times \mathbb{W}$. The out of plane state-transition arcs, $\Delta w$, are computed from $s$ and $s'$. $w$ refers to the task state.*

control policy. We forego the deliberate-and-precise foot placement that is common to humanoid robot control and demonstrate an alternative approach for achieving locomotion tasks in a robust and purposeful fashion.

## 3 Actions

Developing compact models for the action space is crucial for scaling reinforcement learning to high-dimensional settings. For this reason we work with a discrete action space and make decisions at the granularity of character steps. Each action consists of a low-level locomotion controller, such as one step of a walk or run cycle.

For our implementation, we use locomotion controllers of the type proposed by [Yin et al. 2007]. These controllers track target joint trajectories using proportional-derivative control. Balance strategies are incorporated by having the torso and swing hip track desired trajectories in a world frame, and also by using continuous feedback that adjusts the placement of the swing foot. The action vocabulary consists of either predesigned controllers, or intermediate controllers, which are defined by interpolating the torques output by two other controllers. We later provide guidelines for the design of controllers to be used in solving a task (§5).

While the controllers are constructed with the help of a target motion, they generally do not rapidly bring the character state onto a specific motion trajectory. As such, they behave in ways that are qualitatively different from high-gain trajectory tracking controllers. For example, the same controller can be exploited for different purposes in different situations. A forwards walking controller that is invoked from a backwards walking state may be used to induce a rapid stop, while the same motion invoked from an in-place walk will induce forwards motion. This is illustrated in Figure 3. In such situations, the controller is best abstractly characterized as implementing an acceleration action, and not as tighly tracking a specific target motion. The impact of the controller behavior on the modeling of the dynamics is further examined in §6.

## 4 Policy Synthesis

Given a vocabulary of actions $A : \{a^i\}$, the control policy needs to decide which action should be used every time the character takes a step. It does this as a function of the continuously-valued character state, $s \in \mathbb{S}$, and task state, $w \in \mathbb{W}$. The character state $s$ consists of the positions and velocities of the degrees of freedom of the character as seen in its own coordinate frame. Formally, $s = (p, v, q_0, \omega_0, q_1, \omega_1, ..., q_n, \omega_n)$, where $p, v, q_0, \omega_0$ are the root position, velocity, orientation and angular velocity, and $q_i, \omega_i$ are the relative orientation and angular velocity of joint $i$. The orientations are represented by quaternions. The task state $w$ is used to parameterize the task. For example, it could be used to represent the relative position and orientation of the goal with respect to the character. We also define a *global state* $g = (s, w)$ and its corresponding space $\mathbb{G} : \mathbb{S} \times \mathbb{W}$. Formally, the control policy is defined as a mapping $\pi : \mathbb{G} \rightarrow A$, or, equivalently, $a = \pi(g)$.

Policy synthesis begins by modeling the dynamics of the character state in the *state exploration* stage. It is impractical to create a sample-based model for the dynamics that encompasses all possible states because of the high-dimensional nature of the character state. Instead, we focus on modeling the dynamics in a limited region of interest. This is defined with the help of a set of *trusted states* which are treated as starting points for a structured, constrained exploration of the state space. We use a tree-structured exploration process using the set of available actions. This computes an approximate closure for the trusted states under the set of available actions. The character dynamics is modeled as the state transitions encountered during this process, which are captured as a set of dynamics tuples, $D : \{(s, a, s')\}$. Each tuple represents one character step in the context of our locomotion-based tasks, and records the starting state of the character $s$, the applied action $a$ and the resulting state $s'$. We find $s'$ by running a forward-dynamics simulation using the low-level controller associated with $a$ until the next footstrike. An abstraction of the recorded state-to-state dynamics is shown in Figure 4 (left).

Given the dynamics model, the control policy is computed by instancing the character dynamics at multiple sample points in the task space and applying an optimization procedure, *fitted value iteration*, to compute the best global policy. We now describe state exploration, instancing the character dynamics, and fitted value iteration in more detail.

### 4.1 State Exploration

State exploration samples the character dynamics by beginning at a given set of trusted states $T$ and recursively exploring the application of all possible actions, as described in Algorithm 1. With each iteration we grow the set of sampled character dynamics tuples, $D$, by adding new state-action-result tuples $(s, a, s')$, if they meet two criteria. First, the resulting state $s'$ needs to remain sufficiently close to one of the trusted states. This constraint serves as a simple way of modeling what natural poses during a motion are expected to look like and it helps focus our resources on regions of state space that are likely to be important. Second, further recursion is rejected if $s'$ is not considered to be a novel state, i.e., it is within a threshold distance of an existing state. This prevents revisiting an already-explored area. The *trusted* and *novelState* functions both make use of the same character-specific state distance metric, $\mathrm{d}(s_a, s_b)$. For state $s'$ to be considered trustworthy requires $\mathrm{d}(s', s) < \epsilon_T$ for at least one state $s \in T$. A novel state $s'$ is defined as one which satisifies $\mathrm{d}(s', s) > \epsilon_N$ for all states $s \in D$. The

**Algorithm 1** State Exploration

**input** $Q$: queue of states
**input** $A = \{a^i\}$: set of actions
**input** $T$: set of trusted states
**output** $D = \{(s, a, s')\}$: set of dynamics transition tuples
enqueue$(s, \forall s \in T)$
**while** $s \leftarrow$ dequeue() **do**
  **for all** $a^i \in A$ **do**
    $s' =$ forward_dynamics_simulation$(s, a^i)$
    **if** trusted$(s')$ **then**
      $D = D \cup \{(s, a^i, s')\}$
      **if** novelState$(s')$ **then**
        enqueue$(s')$
      **end if**
    **end if**
  **end for**
**end while**

**Algorithm 2** Fitted Value Iteration

**input** $A = \{a^i\}$: set of abstract actions
**input** $D = \{(g, a, g')\}$: global state dynamics tuples
**input** $R(g, a)$: reward function
**output** $\pi^*(g)$: control policy over $\mathbb{S} \times \mathbb{W}$
**output** $V(g)$: value function over $\mathbb{S} \times \mathbb{W}$
$V(g) = 0$ for all $g \in \mathbb{G}$
**while** not converged **do**
  **for all** $g \in \mathbb{G}$ **do**
    $a^* = \underset{a \in A}{\operatorname{argmax}} \big(R(g, a) + \gamma V(g')\big)$
    $\pi^*(g) = a^*$
    $\tilde{V}(g) = R(g, a^*) + \gamma V(g')$
    $V(g) \leftarrow \alpha \tilde{V}(g) + (1 - \alpha) V(g)$
  **end for**
**end while**

character state distance metric is defined by:

$$\mathrm{d}(s_a, s_b) = w_v |v_{s_a} - v_{s_b}| + \sum_{i=0}^{n} w_{\omega_i} |\omega_{i,s_a} - \omega_{i,s_b}| + \sum_{i=1}^{n} w_{q_i} \mathrm{dq}(q_{i,s_a}, q_{i,s_b}),$$

where $\mathrm{dq}(q_a, q_b)$ is the rotation angle, in radians, represented by the quaternion $q_a^{-1} q_b$. The weights used are character specific and are defined in Table 1.

### 4.2 Instancing Dynamics

Figure 4 (right) provides an abstract illustration of how the sampled dynamics is conceptually instanced at multiple points in the task space to produce a dynamics model that spans both the character state and the task state. As with motion graphs, the dynamics of the character state is treated as being invariant with respect to the $(x, z)$ position (where $y$ is up) and the facing orientation of the character in the world. We adopt an approach similar to that of [Treuille et al. 2007] and [Lo and Zwicker 2008] and use a set of task state sample points, $\{w_j\}$ that cover the space of possible goal positions and/or orientations as required by specific tasks. During instancing, the effect of each action on the task state, $\Delta w$, is computed from $s$ and $s'$ for every dynamics tuple $d$, as illustrated in Figure 4. The result of this operation is a set of augmented dynamics tuples $(s, w, a, s', w') \equiv (g, a, g')$.

### 4.3 Fitted Value Iteration

We compute the optimized control policy using a reinforcement learning framework [Sutton and Barto 1998]. Tasks are specified via a reward function $R(g, a)$. The reward function measures the immediate benefit of the character taking action $a$ when at state $g$. The goal of the optimal control policy is to maximize the cumulative long term reward, $V(g) = \sum_{t=0}^{\infty} \gamma^t R(t)$ where $\gamma \in (0, 1)$ is a discount factor that ensures a finite cumulative reward over an infinite planning horizon. The optimal value function can be written in the recursive form given by the Bellman equation,

$$V^*(g) = \max_a \big(R(g, a) + \gamma V^*(g')\big),$$

where $g' = (s', w')$ represents the character and task states which result from applying action $a$ at $g$. For any given global state $g$ the optimal policy, $\pi^*(g)$, is given by the action that maximizes $V^*(g)$.

Estimations of the value function $v_{ij}$ are stored at each sample point $g_{ij}$, corresponding to the instancing of state $s_i$ at the task state point $w_j$. After initializing all $v_{ij}$ to zero, fitted value iteration (FVI) [Ernst et al. 2005] works by iteratively computing improved value function estimates for all $g_{ij}$. The resulting $(g_{ij}, v_{ij})$ tuples are then used to define the new value function $V(g)$ using locally-weighted interpolation. Algorithm 2 provides a full description of the process. We use a form of FVI that follows the core idea of the $TD(0)$ temporal difference learning method with learning rate $\alpha$. We update the value function in-place, which simplifies our implementation, but we could equivalently use other variants of FVI such as batch-mode FVI [Ernst et al. 2005; Lo and Zwicker 2008]. We have experimented with varying the learning rate, $\alpha$, and the resulting policy is largely unaffected although overly small values of $\alpha$ can dramatically slow the convergence rate. In practice, we use $\alpha = 0.8$.

Due to the step-to-step nature of our planning process, by default the discounting of the rewards would be step-based rather than time-based. As a result, the character aims to reach the goal in the fewest number of steps rather than in the shortest amount of time. We can alter this behavior by making the discount rate be a function of $\Delta T$, the duration of a locomotion step. To this end, we use $\gamma(\Delta T) = \gamma^{\Delta T}$.

Given a query point $g_q = (s_q, w_q)$ we estimate $V(g_q)$ from a set of sample points, $\{(g_{ij}, v_{ij})\}$, using a mix of $k$NN and multilinear interpolation. We first identify the set of $k$ nearest neighbors $\{\hat{s}_i\}$, among all the sampled character states. For each $\hat{s}_i$, we use multilinear interpolation to obtain an estimate $V(\hat{s}_i, w_q)$. Finally, we perform $k$NN interpolation among these values to obtain $V(g_q) = \sum_i f(\mathrm{d}(\hat{s}_i, s_q)) V(\hat{s}_i, w_q)$. In the abstract view shown in Figure 4, this corresponds to first interpolating onto the plane corresponding to $w_q$ for all $k$ neighboring states in $\mathbb{S}$, and then interpolating within this new plane for the query character state, $s_q$, using $k$NN interpolation. We use $k = 6$ and a weighting kernel defined by $f = 1/\mathrm{d}^2$. Since FVI repeatedly needs to evaluate the value function at the states $s'$ found in $D$, we precompute and store their $k$ nearest neighbors.

In addition to storing value function estimates, we also store the optimal action $\pi^*(g_{ij})$, computed for each sample point $g_{ij}$. At run time, for a query point $g_q = (s_q, w_q)$, the control policy selects the action associated with $g_{ij}$ where $s_i$ is the character state closest to $s_q$ and $w_j$ is the task state closest to $w_q$. We use this form of nearest-neighbors rather than a weighted interpolation when computing the desired actions in order to remain consistent with a discrete action model. However, given compatible action representations, we speculate that interpolation would also likely produce good results.

| | humanoid | | bird | | raptor | |
|---|---|---|---|---|---|---|
| | $w_\omega$ | $w_q$ | $w_\omega$ | $w_q$ | $w_\omega$ | $w_q$ |
| torso-neck | — | | 1 | 1.5 | 1 | 1.5 |
| pelvis-torso | 1 | 1.5 | — | | — | |
| hips | 0.5 | 1.5 | 0.1 | 1.5 | 0.1 | 1.5 |
| knees | 0.2 | 1 | 0.05 | 1 | 0.05 | 1 |
| neck-head | 0 | 0 | 0.05 | 1.5 | 0.05 | 1.5 |
| body-tail | — | | — | | 0.05 | 1 |

**Table 1:** *Character-specific weights used in the state distance metric. The weights for all other joints are zero.*

## 5 Results

This section is best read in conjunction with viewing the animation results presented in the accompanying video. An executable demonstration of the heading-and-speed task in a physics-based game-like setting is also included with the supplementary material.

**Implementation:** We use the Open Dynamics Engine physics engine [ODE ] on a 2.4 GHz Intel Core 2 Duo with a simulation time step of 0.0005 $s$. All of our results run faster than real time, meaning that in one wall-clock second we can advance the simulation time by more than one second: 2.5× for the bird character, 1.5× for the humanoid and 1.1× for the raptor. The humanoid character has a total mass of 70.4 *kg* and has 34 degrees of freedom. The bird character has a total mass of 64.1 *kg* and a total of 24 degrees of freedom. The raptor character has a total mass of 77.8 *kg* and 42 degrees of freedom, many of which are in the tail. The balls thrown at the characters have a mass of 7—15 *kg*. The weights for the character-specific distance metric are listed in Table 1 and are set with the help of knowing the mass distribution of the character. For example, the big bird has a heavy head relative to the rest of its body and light-weight legs, so the weights used for the related joints reflect this. The head of the humanoid is small and unlikely to have much of an influence on the overall dynamics of the character and thus we set the weights on the related joints to zero in this case.

**Controller design:** The individual controllers used as abstract actions are based on an adapted version of the controllers proposed in [Yin et al. 2007]. They have two underlying states, left-stance and right-stance. Within each state, the target trajectories are represented using Catmull-Rom splines, which we manually edit using a graphical interface (Figure 5) to obtain the variety of controllers necessary for each task. We do not provide the full specification of the exact spline curves here because of the large number of controllers that we use in the various tasks and because the resulting motions are also dependent on other parameters such as the exact geometry and mass distribution used for the characters.

Designing basic low-level controllers is not hard given the right tools and a basic in-place stepping or walking controller to use as a starting point. We have had new users design good running controllers in less than an hour in this way. However, it takes more time to design very natural looking motions and these are often less robust. The final control policy will in the end only be as natural as the underlying controllers. In designing the controllers, the general goal is to span the range of desired gaits needed for the task while also testing to ensure that a variety of reasonable transitions are possible between the controllers. The control policy synthesis method is demonstrably forgiving to limited-robustness controllers that only support transitions in limited situations (§6).

**Trusted states:** The trusted states are chosen to represent the steady-state operation of the basic actions, as well as to encompass natural-looking states obtained when transitioning between different locomotion modes. The set of trusted states is initialized by sampling the limit-cycle states for several of the actions, i.e., forward run, forward walk, in-place walk, backwards walk. Further trusted states are identified with the help of a small number of manually-generated action sequences that are simulated and then observed to see if they yield subjectively 'good' motions, i.e., no falls or other undesirable artifacts. We typically use five sequences of five actions each, which provides a coarse sampling of the kinds of motions that can be generated without needing to run long or exhaustive exploration. For each state $s$ in a good sequence, we test to ensure that *trusted(s)* evaluates to *true*; if this is not the case, we add $s$ to the set of trusted states.

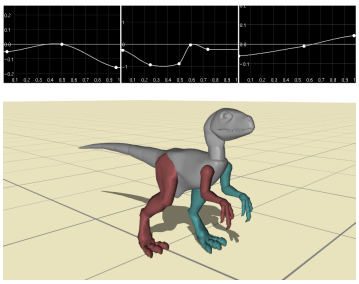


**Figure 5:** *Graphical user interface used to author the low-level locomotion controllers.*

| | $\epsilon_T$ | $\epsilon_N$ | $\lvert T\rvert$ | $\lvert s\rvert$ | FVI time |
|---|---|---|---|---|---|
| Bird GLT | 3.75 | 0.15 | 7 | 2494 | 100 $s$ |
| Humanoid GLT | 2.0 | 0.3 | 24 | 2253 | 104 $s$ |
| Bird HT | 1.5 | 0.3 | 16 | 1915 | 92 $s$ |
| Raptor HT | 2.0 | 0.2 | 60 | 2177 | 86 $s$ |
| Bird GPT | 1.5 | 0.4 | 50 | 4375 | 5 $h$ |
| Bird GPHT | 1.5 | 0.3 | 21 | 1963 | 9.5 $h$ |
| Humanoid VRW | 6.0 | 1.5 | 30 | 1710 | 10 $s$ |

**Table 2:** *Constants and numerical results for the various tasks. Here, $\lvert s\rvert$ indicates the number of states sampled during the state exploration stage using the specified values for $\epsilon_T$ and $\epsilon_N$ and a total of $\lvert T\rvert$ trusted states. The last column indicates the total time required for fitted value iteration.*

**Policy computation:** Details pertaining to the various tasks are presented in Table 2. For all our experiments, the state exploration stage takes between 2.5 and 4 hours, and the FVI converges in 50–70 iterations. The control policies require 0.5—2.5 Mb to store.

### 5.1 Go-to-line task (GLT)

The goal is to reach a designated line by walking forwards, running forwards, or walking backwards towards it. The task state is one-dimensional and represents the position of the line relative to the character. To compute the control policy, we use a total of 35 task state sample points, $-2.5m \leq t_j \leq 6m$, sampled more densely around the origin where extra precision is required for stopping at the right point. The change in task state, $\Delta w$ as shown in Figure 4, is given by the change in root position between $s$ and $s'$, projected onto the sagittal-plane.

The reward function for this task is simple. A reward of 1 is given when the character is within $10cm$ of the goal and has a near-zero forward velocity. A reward of $-1$ is given when the character is in a non-trusted state. In all other cases, the reward is 0. For the bird character, a set of 10 actions is used: backwards run, inplace walk, forward walk and forward run, as well as 6 intermediate actions obtained through interpolation. For the humanoid, we use a similar

set of actions to which we add three actions that help improve the visual quality of the transitions between walks and run.

### 5.2 Heading task (HT)

For this task, the character's goal is to be walking forward in a specified direction. The task state is one dimesional, consisting of the desired heading direction, $\theta$, as measured relative to the current character heading. We compute the control policy over $-\pi < \theta \leq \pi$, and sample at 25 points in this range, sampled more densely around zero. The difference in heading directions between $s$ and $s'$ defines $\Delta w$.

The reward function is again very simple. If the character is walking forward at steady state within 5 degrees of the specified direction, the reward is 1. If the character state is outside the trust region the reward is $-1$, and otherwise it is 0. A set of 32 actions is used by the bird character. To develop these, we designed an in-place walk, a forward walk and two clock-wise turning controllers. The turning controllers are mirrored to achieve counter-clockwise turning. The rest of the actions represent various intermediate controllers. A similar set of actions was used for the raptor controllers. However, we use fewer intermediate controllers resulting in a total of 15 actions.

### 5.3 Go-to-point task (GPT)

A more complex task involves the bird character moving to a specified goal location anywhere on the ground plane. A 2-dimensional task state, $(x, z)$, is used to represent the location of the target relative to the character's frame of reference. We use a total of 729 task states sampled along a $6m \times 6m$ axis-aligned grid with spatially varying density to allow for denser sampling near the goal location, $(0, 0)$. $\Delta w$ is set to the ground-projected difference in root positions between $s$ and $s'$.

We augment the set of actions for the heading task with a backwards walk, a side-stepping motion (mirrored left and right) and one additional interpolated action. This yields a total of 36 actions. Similar to the go-to-line task, the reward function returns 1 when the character is walking in place within 10 $cm$ of the goal, $-1$ when the character's state is too far from the trusted states and 0 otherwise. As demonstrated in the video, this simple reward scheme reveals an unexpected behavior: the bird character sometimes turns and sidesteps in order to quickly stop at the target. To eliminate this behavior, we add a term to the reward function that penalizes large changes in orientation when within $1.5m$ of the target. This demonstrates a simple use of the reward function to help shape the solutions produced by the control policy.

### 5.4 Go-to-point-with-heading task (GPHT)

This task combines the go-to-point and heading tasks. Here, the goal is to be walking at steady-state through a target position while facing a specified orientation. This makes it particularly useful for navigation in environments, as smooth motion paths can be specified using a sparse number of such goals that act as way-points. Specifying a desired heading can also be used to eliminate the unexpected behavior experienced in the go-to-point task. Figure 6 shows a resulting motion.

For this task, we use a three-dimensional task state, $(x, z, \theta)$. The positions are sampled using a $16 \times 16$ grid spanning a $6m \times 6m$ area, and 8 samples are used to sample the orientation dimension, for a total of 2048 task state sample points. The sampling is denser around the goal position and orientation. The value of $\Delta w$ is obtained by combining the analogous components described for the two previous tasks. We use a total of 36 actions, which include those of the heading task, augmented by a backwards walk, left-and-right side-stepping motions, and one additional interpolated action. The reward function is 1 when the goal is satisfied, $-1$ when the state is non-trusted, and 0 otherwise.

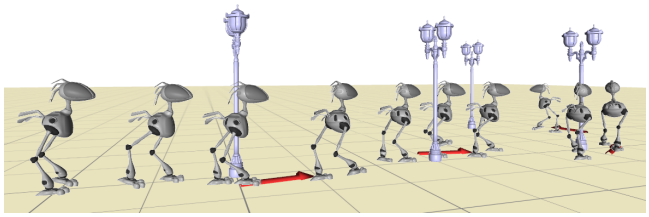


**Figure 6:** *Smoothly walking around lamp posts by walking to goal points with headings.*

### 5.5 Heading with Speed Task (HST)

We build a control policy for following a given heading at a given speed by building on the results of the state exploration for the heading task. The task space is sampled using $5 \times 25 = 125$ states, corresponding to the speed and heading directions, respectively. States that approximately satisfy the desired speed in the heading task are rewarded. The supplemental material includes an interactive game-like demonstration where the player can steer a character through a dynamic, obstacle-filled environment by controlling the character using the desired heading and speed.

### 5.6 Very Robust Walk Task (VRWT)

We generate a particularly robust steady-state walking gait for the humanoid by adding a set of trusted states that correspond to the state at the next foot contact after being hit by $10kg$ and $15kg$ balls. The task is simply to walk at steady state, and so there is no task sampling. The reward is 1 for being at (or near to) the steady state, and 0 otherwise. For actions, we use the in-place walk, forward walk, backwards walk, two modified inplace walks (one leaning slightly forwards and another leaning slightly backwards), and several other interpolated controllers, for a total of 13 actions.

We compare the resulting task control policy to the forward walk controller. For $5kg$ balls thrown at the character at different points in the walk cycle and from random directions, the failure rate with the computed policy drops from approximately 30% to 0%. For $10kg$ and $15kg$ balls, the change in failure rate is similarly $70\% \rightarrow 10\%$ and $100\% \rightarrow 10\%$, respectively.

## 6 Discussion

**Computed vs hand-designed policies:** It is informative to compare the computed policies to carefully hand-crafted policies. For tasks such as the three-dimensional GPH task, hand-designing a good control policy is impractical because of the complexity. Thus, a first benefit of the computed policies is their ease of design using a simple reward function. Even for simpler tasks, optimized policies are not easy to approximate by hand, as shown in the video. Figure 7 visualizes a computed control policy for some of the states from the humanoid go-to-line task. The best actions are both character-state dependent and task dependent. Another difficulty of hand-crafting policies is that the outcome of an action is dependent on the initial state. For comparison, we also develop a carefully hand-tuned controller for the bird heading-with-speed

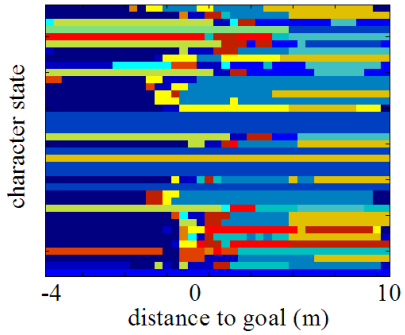



**Figure 7:** *Partial visualization of the humanoid go-to-line control policy. The rows correspond to a random selection of character states. The columns correspond to different distances of the character to the line. Actions are assigned an arbitrary color.*

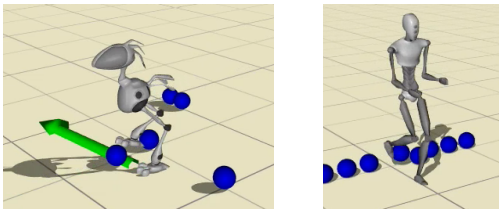


**Figure 8:** *Physics-based interaction with the environment can happen at any time during the task.*

task and quantitatively compare its mean performance to that of the optimized policy. Both situations are evaluated from a regularly sampled set of initial task states. The optimized control policy outperforms the hand-designed policy by up to 32% for some states and a mean difference of 17 % in terms of time-to-goal. This is computed over 96 initial states and ignores the cases where the hand-tuned controller fails outright. Optimized control policies also avoid failure-prone transitions between controllers, as we discuss next.

**Robustness:** A primary motivation for physics-based character animation is that the motions can respond in meaningful ways to a variety of physical interactions with the environment. We perturb the character's motions with heavy balls and a variety of objects that can be stepped on or tripped over, examples of which are shown in Figure 8. We are not aware of other demonstrations of simulated bipedal characters or humanoid robots that have this type of robustness during task execution.

While the underlying controllers are already robust to a certain extent, the task control policy adds robustness in two respects. First, task policies anticipate and avoid the application of actions that lead into unviable areas of the character state space, i.e., where failure is inevitable. The controllers themselves provide no safeguards with respect to being invoked in an inappropriate state. Examples of controller sequences that lead to failure are shown in the accompanying video. A simple prototypical case is that of the turning controllers used for the raptor heading task which are quite sensitive to the initial character state and can result in an awkward side-stepping behaviour leading to an eventual fall.

The computed control policies are also more robust to external perturbations than the individual controllers. For example, a walking controller can receive a perturbation, which, if the walking controller remained active for the subsequent steps would lead to failure. With the use of a task-based conrol policy, a similar state may have been seen during the state exploration phase and as a result, the control policy is well adapted to recover from this state if this can be achieved through the application of another action. Put simply, the task policy combines the strengths of the individual controllers. We refer the reader to the VRW task and examples shown in the video. We note that while the control policies act on a step-by-step basis, the recovery from a perturbtation does not need to happen in a single step. The key is that the character state at any point should never stray too far from example states that have been observed during the state exploration phase.

**Graph-based models of motion:** Control policies can be simpler to develop if the dynamics of the motion can be modeled using a motion graph, wherein there are a discrete set of character states connected by particular available motions. With such a motion model, the character state is assumed to evolve in a predictable and highly constrained way, as given by the discrete choice of paths through the graph. A graph abstraction is used by recent developments that build kinematic control policies based on step-based motion clips [Treuille et al. 2007; Lo and Zwicker 2008]. A blending model is used to allow for transitions between all pairs of motion clips in any given step. In effect, this acts as a funnel that always achieves the end-state of the applied action in one step, irrespective of the starting state. The number of actions and the number of possible character states are equal to the number of motion clips. A related dynamics model is also developed for the Honda ASIMO robot in [Chestnutt et al. 2005], where the history of the last two actions is used as a proxy for the dynamic state of the robot. When used with a set of 7 discrete actions, this yields a motion graph with 49 unique states and $49 \times 7$ edges. The actions thus act as a funnel that achieves a fully predictable end-state after two steps.

The state exploration phase of our method can also be thought of as forming a type of motion graph if we consider state-exploration process as reconnecting to existing motions whenever it passes sufficiently close. The *novelState* function as used in Algorithm 1 effectively models this condition when it is used to help prevent revisiting explored areas of state space. However, this graph has significantly different properties from the graphs described above. The humanoid go-to-line task uses 16 controllers and the exploration yields a total of 4806 unique states, which is significantly larger than for the graph-based methods described above. The number of viable actions per state ranges from 0—14, with a mean of 8 viable actions. This stands in contrast to previous graph-based methods, which generally assume that all actions are viable in all states. The humanoid go-to-line task requires a mean of 6 successive actions to begin at a trusted state and then either fail the trusted-state condition or to approximately reach a repeating state. This contrasts sharply with previous graph-based models of motion which assume that the state history can be fully forgotten after only one or two successive actions. In summary, the actions used in our framework do not behave in the same way that is assumed in prior work on kinematic control policies.

A last notable difference between our work and that of graph-based dynamical models is that our policy and underlying value function are defined over a continuous character state-space, which is not true of graph-based models. The control actions at each step are computed exclusively as a function of the actual current character state. In this respect the method differs from classical tracking approaches; there is no notion of a continuous target trajectory that is always being tracked. A trajectory tracking model needs to know when to stop tracking a given trajectory and jump to a new trajectory, an idea that is explored in part in [Sok et al. 2007]. In this sense, the step-by-step decisions of the task control policy can be seen as jumping to a new trajectory or controller at every step. We speculate that less-robust low-level controllers could be used in our framework, but at the expense of having the high-level control

policy do more of the work. As noted earlier, both the high-level control policy and the low-level controllers contribute to the final robustness of our controllers.

**Dimensionality:** A core challenge in applying reinforcement learning techniques to physics-based characters is the high-dimensional nature of the value function, which needs to span the cross-product of the character state and the task state. We currently rely on two important pieces of *a priori* knowledge to deal with the high-dimension of the state space. Some knowledge of the expected motions is used to seed the state space with a set of trusted states, and thereby help focus the use of resources on modeling the dynamics and control in task-relevant regions of the state space. We also define a character-state distance metric that is used at several points in our pipeline. Good distance metrics can provide meaningful interpolation behavior while avoiding some of the complications that are associated with explicitly modeling low-dimensional embeddings. In future work it would be interesting to automatically learn the best distance metric to use.

Multiple levels of abstraction also contribute towards tackling the high-dimensional nature of the control policies. For example, four levels of abstraction are used in the bird-mania game: (1) the player commands a desired heading-and-speed; (2) the heading-and-speed task policy commands an individual controller; (3) an individual controller commands a set of joint target angles; and (4) joint PD-controllers command joint torques. When seen as a whole, these layers of control implement a mapping from desired heading-and-speed to joint torques. However, directly learning such a mapping is much more difficult than learning level 2 alone, i.e., the task policy.

**Scalability:** A number of factors affect the run-time complexity and storage costs of the method: the dimensionality and sampling of the task space, the size of the action vocabulary, the maximum trusted state distance ($\epsilon_T$), and the minimum novel state distance ($\epsilon_N$). The dimensionality of the task space is perhaps the most important – the $2D$ and $3D$ tasks (GPT, HST, GPHT) require hours to compute the final policy, as compared to the minutes required for the $1D$ tasks. This is because the number of task-space sample states grows exponentially with the task space dimension. We speculate that many tasks which are nominally high dimensional can often still be tackled using lower-dimensional policies. For example, a thrown ball may have a 6D state (3D position and velocity) relative to a character, but can be caught by running quickly to a good intercept point. In this way, additional task and action abstraction may mitigate the complexity of apparently high-dimensional tasks.

The complexity of the character state dynamics does not vary significantly across our examples. All use an action vocabulary consisting of 10–36 controllers and the resulting state dynamics is modeled using 1700–4800 sampled states. This is partly by design – if the set of trusted states and choice of $\epsilon_T$ allow for an expansive exploration of irrelevant regions of the state space, then modeling the state dynamics can become prohibitively expensive.

**Parameter settings:** In order to better understand the sensitivity of the computed task policies to the character state sampling and the task state sampling, we look at how the task policy changes as a function of the sampling for the humanoid go-to-line task. The results are given in Table 3. The quality of the solution generally worsens as we decrease the state sampling and the task state sampling. We have also evaluated the quality by measuring the percentage of actions that remain the same as the optimal choice of action (assumed to be given by the highest sampling) as the sampling is decreased. This decreases monotonically as we decrease either the state sampling or the task state sampling. Sparse sampling may also result in discrepancies between the modeled value function and the rewards encountered during actual policy execution.

| $\|s\|$ | $\|w\|$ | | | | |
|---|---|---|---|---|---|
| | 81 | 57 | 32 | 25 | 17 |
| 4806 | 0 | 0.22 | 0.22 | 0.23 | 0.34 |
| 4000 | 0.18 | 0.35 | 0.28 | 0.25 | 0.35 |
| 3000 | 0.45 | 0.58 | 0.54 | 0.42 | 0.40 |
| 2000 | 0.69 | 0.86 | 0.80 | 0.65 | 0.50 |
| 1000 | 0.79 | 0.99 | 0.87 | 0.75 | 0.60 |

**Table 3:** *Effect of character state sampling and task state sampling for the humanoid go-to-line task. The table gives the mean value function error and uses the most densely sampled case as a baseline.*

As another test, we varied the number of actions used for the same task. We use the same baseline case, i.e., $|s| = 4806$, $|w| = 81$ and which uses 13 actions. For $\{10, 7, 4\}$ actions, the average increase in the value function error across all sampled states is given by $\{1.58, 3.31, 5.46\}$. Thus, a richer action vocabulary clearly does help in better achieving the task.

**Limitations:** The current method has a number of limitations. For any given task, the discrete set of abstract control actions is restrictive as compared to the more arbitrary actions that are afforded by the low-level dynamics. The control actions also remain step-based, which precludes making task-based adaptations to a motion halfway through a step. As a result of such issues, the motions are still not as agile as we would like them to be. It is not always obvious what set of controllers should be made available to the control policy as abstract actions for a given task. The time required for the state exploration stage currently prohibits the iterative design of low-level controllers and reward functions. While we currently use grid-based sampling for the task state, other adaptive approaches may scale better for complex tasks. We have investigated a variety of tasks, but we do not investigate tasks which demand precise foot placement.

## 7 Conclusions

We introduce a method for synthesizing task-based control policies for physics-based animated characters and which can interact with the environment in significant ways. A novel state exploration algorithm provides a structured, constrained exploration of the character state space. We show that with the help of an appropriate distance metric, it is feasible to develop control policies that span the high-dimensional cross product of the character state and the task state for physics-based characters. The task control policies are shown to be more robust than the individual underlying controllers. Our method is tested on six tasks and three characters. We demonstrate one of our control policies in a physics-based game that is included as supplementary material. Our work further opens the door to robust and agile motion for real-time physics-based characters.

## Acknowledgements

We gratefully acknowledge the financial support of NSERC and FQRNT.

## References

ABE, Y., DA SILVA, M., AND POPOVIĆ, J. 2007. Multiobjective control with frictional contacts. In *Proc. ACM SIGGRAPH/EG Symposium on Computer Animation*, 249–258.

ATKESON, C. G., AND MORIMOTO, J. 2003. Nonparametric representation of policies and value functions: A trajectory-based

approach. In *Advances in Neural Information Processing Systems 15*, 1611–1618.

ATKESON, C. G., AND STEPHENS, B. 2007. Random sampling of states in dynamic programming. In *Proc. Neural Information Processing Systems Conf.*

BYL, K., AND TEDRAKE, R. 2008. Approximate optimal control of the compass gait on rough terrain. In *Proc. IEEE Int'l Conf. on Robotics and Automation*.

CHESTNUTT, J., LAU, M., CHEUNG, K. M., KUFFNER, J., HODGINS, J. K., AND KANADE, T. 2005. Footstep planning for the Honda ASIMO humanoid. In *Proc. IEEE Int'l Conf. on Robotics and Automation*.

CHESTNUTT, J. 2007. *Navigation Planning for Legged Robots*. PhD thesis, Carnegie Mellon University.

CHOI, M., LEE, J., AND SHIN, S. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2, 182–203.

COROS, S., BEAUDOIN, P., YIN, K., AND VAN DE PANNE, M. 2008. Synthesis of constrained walking skills. *ACM Trans. on Graphics (Proc. SIGGRAPH ASIA) 27*, 5, Article 113.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics (Proc. SIGGRAPH) 27*, 3, Article 82.

DA SILVA, M., DURAND, F., AND POPOVIC, J. 2009. Linear Bellman combination for control of character animation. *ACM Trans. on Graphics (Proc. SIGGRAPH) 28*, 3, Article 82.

ERNST, D., GEURTS, P., AND WEHENKEL, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research 6*, 503–556.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proc. ACM SIGGRAPH*, 251–260.

HODGINS, J., WOOTEN, W., BROGAN, D., AND O'BRIEN, J. 1995. Animating human athletics. In *Proc. ACM SIGGRAPH*, 71–78.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. A. 2005. Learning to move autonomously in a hostile world. Tech. Rep. UCB/CSD-05-1395, EECS Department, University of California, Berkeley, Jun.

KAJITA, S., KANEHIRO, F., KANEKO, K., FUJIWARA, K., HARADA, K., YOKOI, K., AND HIRUKAWA, H. 2003. Biped walking pattern generation by using preview control of zero-moment point. In *Proc. IEEE Int'l Conf. on Robotics and Automation*.

KHATIB, O., SENTIS, L., PARK, J., AND WARREN, J. 2004. Whole body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics 1*, 1, 29–43.

LASZLO, J. F., VAN DE PANNE, M., AND FIUME, E. 1996. Limit cycle control and its application to the animation of balancing and walking. In *Proc. ACM SIGGRAPH*, 155–162.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/EG Symposium on Computer Animation*.

LAU, M., AND KUFFNER, J. 2006. Precomputed search trees: Planning for interactive goal-driven animation. In *ACM SIGGRAPH/EG Symposium on Computer Animation*, 299–308.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. *ACM SIGGRAPH/EG Symposium on Computer Animation*, 79–87.

LO, W., AND ZWICKER, M. 2008. Real-time planning for parameterized human motion. In *ACM SIGGRAPH/EG Symposium on Computer Animation*.

MCCANN, J., AND POLLARD, N. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3, Article 6.

MORIMOTO, J., AND ATKESON, C. G. 2007. Learning biped locomotion: Application of poincare-map-based reinforcement leraning. *IEEE Robotics & Automation Magazine 14*, 2, 41–51.

MORIMOTO, J., ATKESON, C. G., ENDO, G., AND CHENG, G. 2007. Improving humanoid locomotive performance with learnt approximated dynamics via guassian processes for regression. In *Proc. IEEE Int'l Conf. on Robotics and Automation*.

MUICO, U., LEE, Y., POPOVIC', J., AND POPOVIC', Z. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3, Article 81.

ODE. Open dynamics engine, `http://www.ode.org/`.

RAIBERT, M. H., AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. In *Proc. ACM SIGGRAPH*, 349–358.

SHARON, D., AND VAN DE PANNE, M. 2005. Synthesis of controllers for stylized planar bipedal walking. In *Proc. IEEE Int'l Conf. on Robotics and Automation*.

SOK, K. W., KIM, M., AND LEE, J. 2007. Simulating biped behaviors from human motion data. *ACM Trans. on Graphics (Proc. SIGGRAPH) 26*, 3, Article 107.

SUTTON, R., AND BARTO, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

TEDRAKE, R., ZHANG, T., AND SEUNG, H. 2004. Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proc. Int'l Conf. on Intelligent Robots and Systems*, vol. 3.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3, Article 7.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. SIMBICON: Simple biped locomotion control. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3, Article 105.

YOSHIDA, E., BELOUSOV, I., ESTEVES, C., AND LAUMOND, J. 2005. Humanoid motion planning for dynamic tasks. In *Humanoid Robots*.

ZHAO, L., AND SAFONOVA, A. 2008. Achieving good connectivity in motion graphs. In *ACM SIGGRAPH/EG Symposium on Computer Animation*.

ZORDAN, V., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 24*, 3, 697–701.