Dynamic Terrain Traversal Skills Using Reinforcement Learning

Xue Bin Peng Glen Berseth Michiel van de Panne*

University of British Columbia

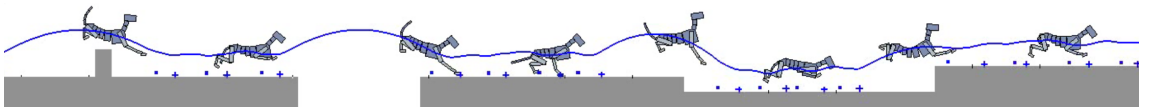

Figure 1: *Real-time planar simulation of a dog capable of traversing terrains with gaps, walls, and steps. The control policy for this skill is computed offline using reinforcement learning. The ground markings indicate the landing points for the front and hind legs.*

Abstract

The locomotion skills developed for physics-based characters most often target flat terrain. However, much of their potential lies with the creation of dynamic, momentum-based motions across more complex terrains. In this paper, we learn controllers that allow simulated characters to traverse terrains with gaps, steps, and walls using highly dynamic gaits. This is achieved using reinforcement learning, with careful attention given to the action representation, non-parametric approximation of both the value function and the policy; epsilon-greedy exploration; and the learning of a good state distance metric. The methods enable a 21-link planar dog and a 7-link planar biped to navigate challenging sequences of terrain using bounding and running gaits. We evaluate the impact of the key features of our skill learning pipeline on the resulting performance.

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

Keywords: Computer Animation, Physics Simulation

1 Introduction

Physics-based simulations arguably offer some of the best prospects for creating models of human and animal motion that generalize well across a wide range of situations. The principle challenge to be solved is that of control, namely determining what actions should be applied over time in order to produce movements that solve a motion task in a natural and robust fashion. Significant progress has been made with regard to motions such as walking, running, and other specific motions, such as falling and rolling. This has been achieved using a variety of control methods including those based on hand-crafted abstractions and feedback laws; policy search to optimize suitable feedback laws; and model-predictive control methods.

In this paper we expand the capabilities of simulated articulated figures by learning control strategies for highly dynamic motions across challenging terrains that contain gaps, walls, and steps. These skills require actions, i.e., jumps and leaps, that are compatible with leaping over (or onto) upcoming terrain obstacles, while also being compatible with the current body state of the character. The interdependence of terrain-obstacle sequences, body-state sequences, and action sequences are a defining characteristic of the problem that needs to be solved. The final control policies should produce actions that are both robust and efficient.

Our solution employs a reinforcement learning (RL) method that is characterized by:

- continuous and high-dimensional states and actions;
- value-iteration based on a set of (s, a, r, s') *transition tuples*, using positive temporal difference (PTD) updates, followed by tuple-culling;
- non-parametric kernel-based value function and policy approximation, with outlier removal;
- an embodied state representation and an optimization process for learning a good state distance metric; and
- a progressive learning approach that alternates between ϵ-greedy exploration and value iteration.

Our contribution lies in showing that, taken together, this set of algorithmic choices opens the door to the development of new classes of skills for simulated characters. The difficulty of applying RL to continuous state and action spaces is well known, and thus our work provides insights into how RL methods can be successfully applied to such problems. We evaluate the method on a simulated 21-link planar dog and a 7-link planar biped that learn to navigate terrains with sequences of gaps, walls, and steps. We investigate the impact of the various design decisions that characterize our approach.

2 Related Work

The use of physics-based models in computer graphics has yielded tremendous dividends in rendering, geometric modeling, and the animation of phenomena such as fluids and cloth. Similar dividends are available for physics-based character animation, although progress in this area has been tempered by the difficulty of modeling the motor control skills of humans and animals that are also required to drive the simulations. This topic has a rich history and we refer the reader to a recent survey paper [Geijtenbeek and Pronost 2012] for an overview.

*e-mail: xbpeng|gberseth|van@cs.ubc.ca

ACM Reference Format
Peng, X., Berseth, G., van de Panne, M. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. ACM Trans. Graph. 34, 4, Article 80 (August 2015), 11 pages. DOI = 10.1145/2766910 http://doi.acm.org/10.1145/2766910.

SIGGRAPH '15 Technical Paper, August 09 – 13, 2015, Los Angeles, CA.

DOI: http://doi.acm.org/10.1145/2766910

Physics-based character animation: One approach for control is to specify short term goals for the motion using a mix of constraints and objectives. The control can then be computed using inverse dynamics (if fully constrained), solving a quadratic program for the current timestep, or by solving a finite-horizon trajectory optimization, e.g., [Stewart and Cremer 1992; Jain et al. 2009; Macchietto et al. 2009; de Lasa et al. 2010; Mordatch et al. 2010] and others. Because these methods leverage an explicit model of the equations of motion (EOM), they are often categorized as model predictive control (MPC) methods. Another category of approach is to develop explicit control policies that directly use high-and-low level goals to compute the actions that help achieve these goals, without assuming full knowledge of the EOM, e.g., [Raibert and Hodgins 1991; Yin et al. 2007; Ye and Liu 2010; Wang et al. 2009; Coros et al. 2010; Coros et al. 2011; Tan et al. 2014]. These methods include the use of foot-placement feedback laws, Jacobian transpose control, inverted pendulum models, and PD-controllers. Offline and online optimization play an important role in many of the methods, along with human insight into design of tasks, controller structure, and objective functions, e.g., [Ha and Liu 2015].

Balance is a key issue to be tackled for achieving robust locomotion, and thus the above approaches generally include a strategy or objective that allows for the anticipation of upcoming footplants. Many methods also rely on motion capture data, which is one way to achieve more realistic motions, e.g., [Zordan and Hodgins 2002; Kwon and Hodgins 2010; Lee et al. 2010b; Coros et al. 2011; Wei et al. 2011]. Some capability for coping with terrain obstacles is incorporated into several control strategies [Liu et al. 2012; Ha et al. 2012; Yin et al. 2008; Coros et al. 2008; Mordatch et al. 2010; Coros et al. 2011]. In comparison to this existing work, the approach we propose offers significantly more capable motions in terms of tightly-sequenced obstacles and types of terrain obstacles, and it develops these capabilities in a principled fashion. In relation to prior work on learning for bicycle stunts [Tan et al. 2014], our work tackles problems that are characterized by obstacles rather than balance regulation, that have high-D outputs (14–28 vs 1–3), and that employ value-iteration instead of policy search.

Reinforcement Learning: Learning good control strategies is an enticing alternative to *designing* them. Reinforcement learning (RL) offers a general approach to this problem. It seeks to learn the best action to take in any given state such that the resulting motions maximize a cumulative reward function. This includes *policy search* methods that optimize directly over a given set of policy parameters. In animation, RL has been applied with great success to kinematic models of motions, where the actions consist of motion clips, with possibly constrained connectivity. At the end of any given motion clip, the control problem is then to choose the next clip to blend and play. This framework has been applied to tasks such as responding to user-directed goals, i.e., "now walk in this direction", moving to a desired goal location, or interacting with other characters [Lee and Lee 2006; Treuille et al. 2007; McCann and Pollard 2007; Lee et al. 2009]. Our work draws inspiration from previous work on *motion fields* [Lee et al. 2010a], which develops kinematic controllers for a given set of transition tuples. We also work with RL as applied to tuple sets, but also support the generation of new states and actions during exploration.

RL methods are challenging to apply to physics-based character animation because the control is more indirect: the motion will inevitably drift away from the intended one, which is not the case when playing motion clips. Furthermore, while motion clips act as discrete states (which clip is currently playing) and discrete actions (which clip should be played next), a physics-based character has a continuous, high-dimensional state space and action space. It is also not obvious in advance which regions of the state space will be visited and which actions will be useful. To make progress on this problem, one simplification has been to use a set of existing locomotion controllers to define a discrete action space [Coros et al. 2009]. At the start of each locomotion step, the control policy selects a controller from N predefined controllers for use during the upcoming step. This has the limitation that the set of actions needs to be fixed in advance, which is problematic in our setting. We wish to have a continuous action space that can be explored as needed in order to deal with novel states that may arise during the learning process.

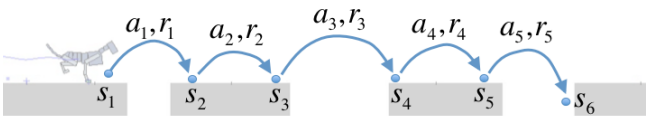

Figure 2: *Control policy for dog on gaps terrain.*

A large body of research is dedicated to reinforcement learning and our review can only touch on a number of representative works due to space constraints. A good discussion of the specific challenges that are presented by *continuous action spaces* is given by van Hasselt et al. [Van Hasselt and Wiering 2007; Van Hasselt 2012]. Policy search methods and non-linear function approximators commonly play key roles in many methods. Several recent approaches rely on algorithms that alternate between collecting potentially better solution samples in the vicinity of a current policy and policy adaptation using the new samples, i.e., [Ross et al. 2011; Levine and Koltun 2014]. Alternatively, value-iteration or policy-iteration methods require the use of function approximators to model the value function, $V(s)$, or relatedly, the Q function, $Q(s, a)$. We refer the reader to a recent monograph [Busoniu et al. 2010] for a general overview of RL with function approximation. While methods based on global basis function approximation are best suited to low-dimensional settings, methods based on non-parametric kernel estimation [Ormoneit and Sen 2002] and Gaussian processes [Engel et al. 2005] may generalize better to high-dimensional settings. Often there is also significant benefit to be gained from *experience replay* or *batch-based* RL methods, an overview of which can be found in [Lange et al. 2012] and [Fonteneau et al. 2013]. As described earlier (§1), the RL methodology we develop in this paper is characterized by a number of key design choices and we evaluate the impact of these choices in our results. Our value iteration method is closest in nature to the CACLA algorithm [Van Hasselt and Wiering 2007; Van Hasselt 2012], that also uses batched positive temporal difference updates, and which is tested on problems with a 1D action space.

3 Overview

Our goal is to generate control policies that enable a physics-based character to traverse given classes of terrain with a dynamic gait. Figure 2 illustrates an example *gaps* terrain, which consists of gaps of random width that are spaced at random intervals. The control policy selects actions as a function of the current state, e.g., $a_1 = \pi(s_1)$, which results in a subsequent state s_2. As illustrated in the example, each action, a_i, produces a complete bound or leap that may succeed or fail, e.g., s_6. Actions begin and end on a specific contact event, for example when the dog's hind legs come into contact with the ground. In deciding on the best action, the control policy needs to consider the current state of the dog, i.e., its particular pose and related velocities, as well as the state of the upcoming terrain. The state, s, thus encompasses both of these factors. Lastly, each action also results in a reward, $r(s, a)$, that rewards forward motion and penalizes falls and movement effort.

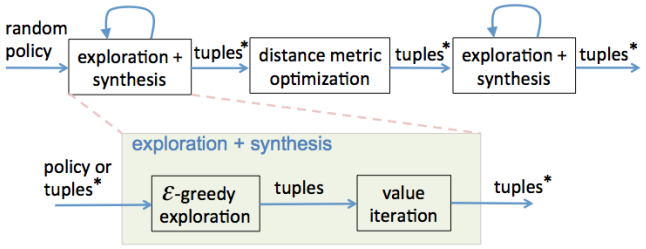

Figure 3: *Overview of skill learning pipeline.*

The multi-step *learning pipeline* that we develop strives to learn an optimal control policy, which is defined as one that given a current state, s, maximizes $V(s)$, the sum of future-discounted rewards that result from the policy decisions. The learning is done in a progressive fashion through the continued accumulation of **transition tuples**, $\mathcal{T} = \{(s_i, a_i, r_i, s_i')\}$. These embody all the past experiences of the character in traversing example terrains during *exploration* stages. The tuples resulting from exploration are added to any existing tuples and are then used in a *value iteration* stage to compute an optimal policy, as shown in Figure 3 (bottom). Importantly, the policy and the related value function are represented using the tuples themselves by augmenting each tuple, T_i, with a *value function* value, V_i, and an *optimal policy tuple* attribute, $\Pi_i = true|false$. These serve to represent the value function and policy action (implicitly a_i if $\Pi_i = true$) at state s_i. The optimal policy tuple set, $\mathcal{T}^*$ is then defined by the set of all tuples that have the attribute Π_i enabled, i.e., $\mathcal{T}^* = \{T_i|\Pi_i = true\}$.

A small set of fixed actions, which are designed by hand or during a separate optimization process, are used to initialize the policy synthesis process. A policy that makes uniform random choices among the given actions is used to collect an initial set of transition tuples that is used to develop an initial policy. Multiple further iterations of exploration and synthesis are then used to produce successive improvements in the control policy, as ilustrated in Figure 3.

A key step in our learning pipeline is *distance metric optimization*. Our non-parametric (i.e., sample-based) function approximators for estimating $\hat{V}(s)$ and $\hat{\pi}(s)$ are based on kNN interpolation. This therefore relies heavily on having a good distance metric to find the most relevant nearby states $\{s_i\}$ for a query state, s. The distance metric optimization is used to replace the initial rudimentary hand-designed distance metric with one that is specifically tailored for the given category of terrain. With the improved distance metric in hand, additional exploration and synthesis stages are used to achieve further policy improvements. Lastly, an optional tuple culling step (not shown in Figure 3), is used to significantly reduce the final number of tuples in the policy.

Throughout the iterations of exploration and synthesis, new actions are explored, which leads to new regions of the state space being visited. This progressive, coupled exploration of the state and action spaces is a significant feature of the method, as it avoids having to predefine bounded domains for states and actions. The use of distance metric optimization also allows for arbitrary features to be used in the state vector and they will be weighted in accordance with their utility.

4 Policy Synthesis

A control policy, $\pi : S \rightarrow A$, is defined by a mapping from the state, $S \in \mathbb{R}^n$, to an action, $A \in \mathbb{R}^m$. The characters whose motion we wish to control have high-dimensional and continuous state spaces and actions spaces. For our problem, the state $s \in \mathcal{S}$ models both the state of the character and the state of the upcoming terrain. Actions are modeled in terms of discrete bounds or leaps, i.e., a single action models the control required to complete a single bound or leap for the dog, and a single running step for the biped. We defer a more specific description of the state and action parameterization until later (§5, §6).

Value function: As is standard in reinforcement learning, we wish to learn an optimal control policy that maximizes the discounted cumulative reward, as represented by the objective:

$$J = \sum_{i=0\ldots\infty} \gamma^i r_i(s_i, a_i),$$

and $\gamma < 1$ is the discount factor. The value function, $V(s)$, models this as a function of the state, and can be expressed compactly as:

$$V(s) = r(s, a) + \gamma V(s')$$

where s is the current state, a is an action defined by a control policy, and s' is the state resulting from the action. The optimal policy, π^* is one that maximizes $V(s)$ and therefore satisfies:

$$V^*(s) = \max_a \{r(s, a) + \gamma V^*(s')\}.$$

or, equivalently,

$$V^*(s) = r(s, \pi^*(s)) + \gamma V^*(s').$$

The value function, $V(s)$, and control policy, $\pi(s)$ are intimately related as shown above. The key challenge, then, is to develop the right representations and algorithms to compute optimal solutions, given the continuous and high-dimensional nature of both states and actions in our problem setting. The method we adopt is based on *value iteration* as implemented via *positive temporal difference* updates applied to *transition tuples*, which we now describe in further detail.

Transition tuples: A set of transition tuples, $\mathcal{T} : \{T_j\}$, forms the basis for computing our control policies where each tuple, $T_j = (s_j, a_j, r_j, s_j')$, describes a transition from a state s_j to a state s_j' after taking an action a_j, and receiving a reward r_j during this transition. Here, j represents an arbitrary tuple index, i.e., not a discrete time index. We drop this index where this can be done without ambiguity, refering to the tuple elements as (s, a, r, s'). The tuples are collected during the exploration phases using episodic simulations across randomly-generated terrains. Each episode consists of a sequence of actions, i.e., dog bounds or biped running steps, that continue to advance the character across the terrain until a fall occurs, at which point another episode is started. The episodes generate contiguous sequences of states, i.e., the ending state, s', of one action becomes the starting state, s, of the next action. However our conceptual view treats these as a set of independent tuples, as illustrated in Figure 4 (a).

Non-Parametric Function Approximation: Given the continuous nature of the states and actions, function approximators are needed to represent the value function, $V(s)$, and the control policy, $a = \pi(s)$. We use a non-parametric approximation method for both, based on the tuples, $\mathcal{T}$. Each tuple, T_i, stores a value function, V_i, and action, a_i, that are associated with the starting state of the transition tuple, s_i, as illustrated in Figure 4 (b). For the function approximation, we use kNN interpolation with a Gaussian kernel:

$$\hat{V}(s) = \frac{1}{W} \sum_{i=1\ldots k} w_i V_{N_i(s)} \delta_i$$

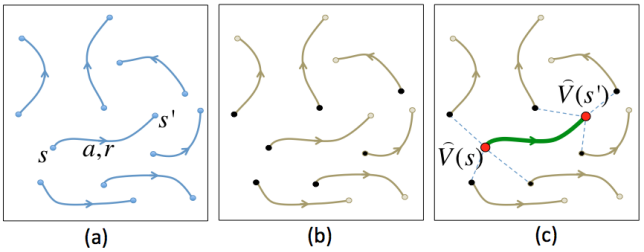

Figure 4: *Transition Tuples. (a) Conceptual view of tuples in state-space. (b) Data points used for non-parametric modeling of $V(s)$ and $\pi(s)$. (c) Points used to compute the value function update, and the neighbors for kNN interpolation of V.*

where $W = \sum_{i=1\ldots k} w_i \delta_i$, $N_i(s)$ returns the tuple index of the ith nearest neighbor of state s, $w_i = k(d) = e^{-d^2/\sigma_\lambda^2}$ and σ_λ is a kernel width. The kernel distance, d, is given by:

$$d(s, s_j) = (s - s_j)^T M (s - s_j),$$

where $M = \text{diag}(\mu)$ and μ is a vector defining a set of to-be-learned normalized weightings for the state vector components. Lastly, δ_i represents a trimming function that can be used for outlier removal:

$$\delta_i = \begin{cases} 0 & : \ T_i \text{ is an outlier with respect to N(s)} \\ 1 & : \ \text{otherwise} \end{cases}$$

For value function approximation, we use $\delta_i = 1$, i.e., we do not apply outlier removal. This allows the value function to be continuous even when the underlying policy has a discontinuity, as motivated by what occurs during optimal control of bang-bang control tasks. However, for the policy function approximation, $a = \pi(s)$, outlier removal is crucial to performance, as we detail shortly. In our implementation, $k = 10$. To facilitate fast queries with large numbers of tuples in a high-dimensional space, we use the FLANN library [Muja and Lowe 2009] with 4 randomized k-d trees.

Outlier removal: The averaging applied by a kNN approximation can be problematic for policy approximation. Figure 5 illustrates an example scenario that leads to a sharp discontinuity in the control policy. The dog can choose to make an additional short bound before leaping across the gap, or can directly leap across the gap. Both actions are well suited to the situation and thus are likely to exist as promising actions for nearby states and therefore play a role in the kNN approximation. However, averaging of these very different actions leads to a poor interpolated outcome, i.e., landing in the middle of the gap. A nearest-neighbor policy approximation avoids this averaging. However, this eliminates averaging in the common case where it is desirable, and can yield poor decisions because of the strong dependence on only one data point. In the case of discrete actions, a voting approach can be used and we thus use this for discrete-action policy approximation for which we shall perform comparative evaluations.

Our solution is to discard outliers, where an outlier is defined as a kNN member whose action differs more than some epsilon from the kNN mean action. Unfortunately, we do not have a good distance metric for the action space. We do, however, have a good distance metric for the state space, which we leverage as follows. Actions are deemed similar if they lead to similar states. Given that the k nearest neighbors were selected based on the proximity of their starting states, s, we use their end-state differences, $s' - \bar{s}'$, as a proxy for their action differences. We therefore identify the kNN tuple whose end state s' is the furthest from the weighted mean of the end states. If this distance is further than a given δ, it is identified as an outlier that should be ignored, and the process is repeated. We use a hand-tuned value of $\delta = 0.05$. The algorithm is summarized in Algorithm 1.

Algorithm 1 Remove outliers

1: **input** s: query state
2: **input** $S = \{s_j\}$: neighbour start states
3: **input** $S' = \{s'_j\}$: neighbour end states
4: **output** J: set of neighbours with outliers removed
5: $\bar{s}' \leftarrow$ **Weighted Average**(s, S, S')
6: $j \leftarrow \arg\max_j d(s'_j, \bar{s}'), s_j \in S'$
7: $d_{max} \leftarrow d(s'_j, \bar{s}')$
8: **while** $d_{max} > \delta$ **do**
9: $\quad S' \leftarrow S' \backslash s'_j$
10: $\quad S \leftarrow S \backslash s_j$
11: $\quad \bar{s}' \leftarrow$ **Weighted Average**(s, S, S')
12: $\quad j \leftarrow \arg\max_j d(s'_j, \bar{s}')$
13: $\quad d_{max} \leftarrow d(s'_j, \bar{s}')$
14: **end while**
15: $J = \{j | s'_j \in S'\}$

Distance metric learning: The weights of the distance metric and the kernel width, σ_λ, are learned via policy search with CMA [Hansen 2006] using the most recently available set of tuples, $\mathcal{T}$. The free parameters for the optimization are defined by $\phi = \{\mu_i\} \cup \sigma_\lambda^2$. The objective is measured by the cumulative performance over a set of $e = 10$ environments:

$$\phi^* = \arg\max_\phi \sum_{i=1}^{e} \text{dist}_i(\phi)$$

where $\text{dist}_i(\phi)$ is the distance traveled on terrain i using the optimal policy computed using the parameters ϕ. In order to avoid noise in the objective function, the e terrains are randomly generated once and then remain fixed.

Value iteration: Given a set of transition tuples, $\mathcal{T}$, obtained using exploration, we use value iteration to compute a policy that is optimal with respect to the observed tuples. While RL problems having discrete actions spaces can employ full backup operations, i.e., they can explicitly maximize over all possible actions, this is not possible in a continuous action setting. We apply iterative batch processing of updates based on positive temporal difference updates, as summarized in Algorithm 2. For each transition tuple, the temporal difference measures the net benefit of taking the tuple transition as compared to the current value function estimate for the state (line 10). For 'winning' tuples having $\delta > 0$, the value function is adapted and the tuple is identified as one that contributes towards the optimal policy. Value iteration terminates when the maximum positive temporal difference seen on the current iteration is

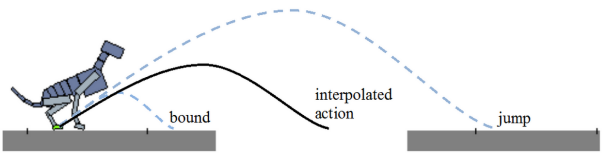

Figure 5: *Undesired averaging of neighboring actions.*

Algorithm 2 Positive temporal difference value iteration

input $\mathcal{T} = \{(s_i, a_i, r_i, s_i')\}$: state transition tuples
input α: learning rate
input γ: discount rate
output v_i: value functions at s_i
output $\Pi_i = \{true|false\}$: optimal policy tuple flag
$v_i \leftarrow 0$ for all i
$\Pi_i \leftarrow false$ for all i
while not converged **do**
  for all T_i **do**
    $\delta = r_i + \gamma \hat{V}(s_i') - \hat{V}(s_i)$
    if $\delta > 0$ **then**
      $v_i \leftarrow v_i + \alpha\delta$
      $\Pi_i \leftarrow true$
    else
      $v_i \leftarrow v_i + \alpha(\hat{V}(s_i) - v_i)$
      $\Pi_i \leftarrow false$
    end if
  end for
end while

less than a given threshold, i.e., $\delta < 0.001$. The final policy is comprised of the actions of the winning tuples, $\mathcal{T}_\pi : \{T_i|\Pi_i = true\}$.

Exploration: Tuples are collected by using ϵ-greedy exploration that is further divided into local exploration and global exploration. In *local exploration*, new actions are constructed by introducing perturbations to locally optimal actions and therefore the new actions will be similar to the actions that are suggested by the policy. The perturbations are sampled from a distribution that favors changes that are correlated with nearby locally optimal actions, as modeled by a multivariate Gaussian. For a given state s, a set of policy tuples, $\mathcal{T}$, the set of locally optimal actions are determined by finding the neighbouring tuples,

$$\mathcal{N} = \{i|T_i \in \mathcal{T}, d(s, s_i) < r_{\max}\}$$

where $r_{\max}$ defines a maximal radius of interest. The correlation matrix, C, is constructed according to:

$$C = D^{-1}\left(\frac{1}{W}\sum_{i\in\mathcal{N}} w(s, s_i)(a_i - \overline{a})(a_i - \overline{a})^T\right)D^{-1}$$

where $W = \sum_{i\in N} w(s, s_i)$, $w(s, s_i)$ are the kNN weights as defined earlier, and $D = diag(\sigma_1, \ldots, \sigma_n)$, and σ_i^2 are the variances of the action vector as computed from all actions in all the policy tuples, $\mathcal{T}^*$. The new action a' is then determined according to:

$$a' = a + D\,(C + \nu I)\,N(0, \sigma).$$

Here, νI acts as a regularizer to ensure that the resulting matrix is full rank and also allows perturbations to move out of the subspace spanned by the current local optimal actions. We use $r_{\max} = 0.2, \nu = 0.1, \sigma = 0.6$.

Actions learned in one region of state space may also be effective in other regions. This motivates a more global form of exploration, in addition to local exploration, to allow for a better transfer of skills between more distant regions of the state space. At each exploration cycle, with a probability α, an action is chosen randomly from all tuples in $\mathcal{T}^*$. The choice of actions at each cycle are summarized as shown in Table 1.

Tuple Reduction: While the final control policy is defined by all the winning tuples, it is possible to produce more compact policies by removing many tuples that make only a negligible contribution. First, some tuples may correspond to actions taken in rarely-visited regions of the state space. For example, the initial random policy will result in visiting states that are never again seen once the policy becomes more skilled. Second, some tuples may be redundant if they correspond to on-policy experiences that repeatedly visit similar areas of the state space. We only investigate the first type of reduction, namely removing rarely-used tuples, as quantified by running the policy for a fixed number of cycles (100k for the dog, 200k for the biped). Any tuples that are not used by the kNN kernel during this policy sampling are then tagged for removal. Other similar strategies for producing a refined or reduced set of sample points would also be possible, such as the approach suggested in [Levine et al. 2012].

Probability	Action
$1 - \epsilon$	exploit policy
$\epsilon(1 - \alpha)$	local exploration
$\epsilon\alpha$	global exploration

Table 1: *ϵ-greedy exploration actions. We use $\epsilon = 0.4, \alpha = 0.3$.*

5 Action Parameterization

Our characters are modeled as articulated figures whose motion is then simulated by applying torques at joints. The planar dog model is composed of 21 links, has a total mass of 32.4 kg, and uses the pelvis as the root link. The planar biped model has 7 links, a total mass of 50.4 kg and uses the torso as its root link. The motion is simulated using the Box2D physics engine [Box2D 2015] at 3000 Hz using a coefficient of friction of 0.8. Torque limits are 500 Nm for the dog and 300 Nm for the biped.

The choice of action parameterization can significantly impact the efficacy of reinforcement learning algorithms in continuous domains. We develop actions using phase-based finite-state machines (FSMs), as is common for locomotion controllers. An action consists of a specific instantiation of the FSM that is selected at the start of each locomotion cycle, and thus specifies a single leap for the dog or a single running step for the biped. Within each motion phase of the FSM, we use a mix of low-level features to control the motion, e.g., desired joint angles, and abstract features, e.g., desired leg forces.

Dog Actions: The basic dog actions are bounds or jumps, which are segmented into four phases, as shown in Figure 6. State transitions occur after a set period of time has elapsed, or after a particular link makes contact with the ground. Upon contact of the hind foot, the last action ends and the new action begins in the *back-stance* phase. Within each phase, control is computed in terms of the joints and features illustrated in Figure 7. The pelvis acts as the root link and the orientation of every link is defined with respect to its parent link, with the exception of the shoulder, hip, and ankles which specify the orientation of their child links in world space. The orientation of each joint is controlled using proportional-derivative (PD) control to track specified joint trajectories. Joints that are not specifically listed as control parameters simply track a fixed default angle using PD control. The joints for the spine are controlled in a simplified and coordinated fashion using a single *curvature* parameter to set all the PD-target angles.

The dog's legs also exert virtual forces that are realized by internal torques computed via the Jacobian transpose. These torques are then added to those supplied by the PD-controllers. The virtual forces for the legs are applied only when their respective foot is in

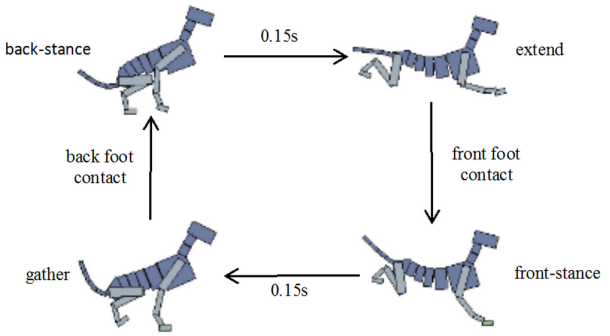

Figure 6: *Control states for the dog.*

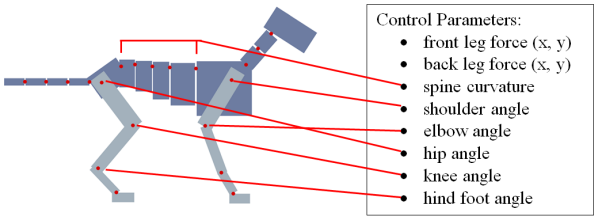

Figure 7: *Control parameters for the dog.*

contact with the ground and only during the appropriate respective phases, i.e., the back leg will exert a virtual force only when the back foot is contacting the ground during the back-stance phase.

In addition to applying a specified constant leg force, which is treated as a free control parameter, each leg also applies gravity compensation for all links supported by the leg. Links comprising the front-half of the body are associated with the front leg and links comprising the back-half of the body are associated with the hind leg. Lastly, virtual force feedback is applied to guide the hip and shoulder joints to a desired height with zero vertical velocity. In total, the force exerted by one of the legs is given by:

$$F_{\text{leg}} = F_0 + c_d(h - h_0) + c_v(\dot{h} - \dot{h}_0) + g\sum_{i \in L} m_i$$

where F_0 is the force specified by the control parameters, h_0 and $\dot{h}_0$ are the desired height and vertical velocity of the shoulder or hip joint, and c_v and c_d are the distance and velocity gains for the respective joint, and L is the set of links supported by the leg. F_{leg} is then realized by applying internal torques, computed with the Jacobian transpose, to the joints of each respective leg. When a foot is not in contact with the ground, a balance feedback strategy based on center-of-mass velocity, i.e., [Raibert and Hodgins 1991], is also applied to adjust the target angle of the shoulder or hip according to $\theta' = \theta_0 + c_v(v_{\text{com}-v_0})$, where $c_v = 0.2$ s rads/m and $v_0 = 4.0$ m/s. In total, the dog has a 28-dimensional action space, as defined by the free parameters associated with each phase of the motion, as listed in Table 2.

Biped Actions: The biped running FSM consists of two states for each step, as shown in Figure 8. The action begins and ends with a foot making contact with the ground to become the new stance foot and transitions to the next state after 0.15s. The second state ends when the swing foot makes contact with the ground to become the new stance foot. The biped has the torso as its root link. Similar to the dog, all joint angles are defined relative to their parent links with the exception of the swing hip joint, which is specified

Action Parameter	Active Phases
spine curvature	1,2,3,4
shoulder angle	1,2,3,4
elbow angle	1,2,3,4
hip angle	1,2,3,4
hock angle	1,2,3,4
hind foot angle	1,2,3,4
hind leg force (x, y)	1,3

Table 2: *Dog Action Parameters. Phases 1,2,3,4 correspond to Back Stance, Extend, Front Stance, and Gather, respectively.*

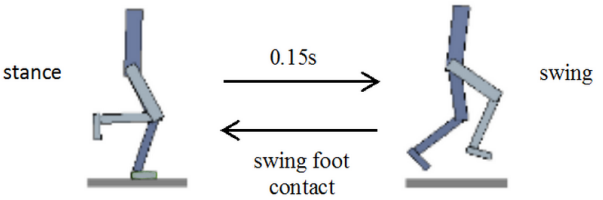

Figure 8: *Biped Finite State Machine.*

in world space. The control parameters provide target angles for all the joints, with the exception of the stance hip joint which is then used to control the torso pitch.

As with the dog, a constant virtual force is applied when the stance foot is in contact with the ground, implemented using the Jacobian transpose. The stance leg also provides a vertical force for gravity compensation. Lastly, foot-placement feedback [Yin et al. 2007] is achieved via the swing hip as a function of the horizontal velocity of the center of mass and its location relative to the stance foot: $\theta' = \theta_0 + c_v(v_{\text{com}} - v_0) + c_d(d_{\text{com}} - d_0)$, where $c_v = 0.2$ s rads/m, $v_0 = 4.0$ m/s, $c_d = 2.2$ rad/m, and $d_0 = 0$ m. The biped has a 16-dimensional action space as defined by the free parameters associated with each phase, listed in Table 3.

Initial Actions: A fixed number of initial actions are provided to initialize the skill-learning pipeline. Since the environments require leaping over or onto obstacles, we generate a set of initial actions, some of which are capable of fixed-speed bounding or running, and others of which are capable of a basic jump or leap. For the dog, the initial actions are comprised of 4 bounds of various speeds and 4 jumps of various distances. The bounds are developed by uniformly-spaced interpolation between a slow bound and the default bound, while the jumps are similarly developed by interpolating between the default bound and a jump.

The default and slow bounds are optimized to maintain a desired speed while minimizing the sum of torques applied to all joints.

Action Parameter	Active Phases
torso pitch	1,2
swing hip	1,2
swing knee	1,2
swing ankle	1,2
stance hip	1,2
stance knee	1,2
stance ankle	1,2
stance leg force (x, y)	1

Table 3: *Biped Action Parameters. Phases 1 and 2 correspond to Stance and Swing, respectively.*

This is captured by minimizing the following objective:

$$\int_0^{t_{\max}} (w_v E_v(t) + w_\tau E_\tau(t))\mathrm{d}t$$

where $w_v = 5$, $E_v(t) = (v_{\mathrm{com}}(t) - v_0)^2$, $w_\tau = 5 \times 10^{-8}$, $E_\tau(t) = \sum_j \tau_j^2(t)$. We use $v_0 = 4.0$ m/s for the default bound and $v_0 = 2.0$ m/s for the slow bound.

The jump is optimized for horizontal distance and effort by minimizing the following objective:

$$\int_0^{t_{\max}} (w_v E_v(t) + w_\tau E_\tau(t))\delta_{\mathrm{jump}}(t)\mathrm{d}t$$

where $w_v = 20$, $E_v(t) = -v_{\mathrm{root}}$, $w_\tau = 1 \times 10^{-9}$, E_τ is defined as above, and

$$\delta_{\mathrm{jump}} = \begin{cases} 1 & : \text{ jump cycle is active} \\ 0 & : \text{ otherwise} \end{cases}$$

The optimization for the initial actions are solved using covariance matrix adaptation (CMA), as implemented by the Shark library [Hansen 2006]. We expect that other derivative-free optimization methods may also do well.

Initial actions are developed for the biped in a similar fashion with the goal of providing reasonable initial running steps and leaps that can then be further adapted. While developing good leap actions, we noted this also benefits significantly from having dedicated anticipatory and recover steps that precede and follow the leap. As such, we initially model the leaps as *extended actions* that consist of an (anticipate, leap, recover) action sequence. A pool of 8 actions is then developed, consisting of: (a) 4 running speeds, as interpolated between a default run (4 m/s) and a slow run (2 m/s); and (b) 4 leaps, as interpolated between a large leap extended action and a no-leap extended action that is simply modeled using the default run action for each of the (anticipate, leap, recover) actions. During the initial random-policy exploration, actions are randomly chosen from this pool of 8 actions. Once this has produced an initial pool of transition tuples to work with, the notion of extended actions is discarded, given that the tuple set then already contains example action sequences for achieving large leaps.

6 Task Descriptions

Terrains: Skills are developed for multiple classes of terrain. Within a given terrain class, the environments are randomly generated by drawing successive samples from uniform distributions over the specified ranges for each feature. The *gaps* terrains for the dogs consists of flat ground interruped by gaps of width $w \in [1, 4]$ m. Gaps have a depth of 2 m and are spaced at a distance $d \in [4.5, 7]$ m apart. The *gaps* terrains for bipeds are similarly defined using $w \in [0.5, 2]$ m and $d \in [4.5, 7]$ m. The *steps* terrains for the dog are defined by step heights $h \in [-0.75, 0.75]$ m spaced at a distance $d \in [7, 10]$ m apart. The *walls* terrains for the dog is composed of flat terrain populated with walls of width $w = 0.5$ m and height $h \in [0.25, 1]$ m that are spaced at a distance $d \in [5, 12]$ m apart. Similarly, the *walls* for the biped are defined by $w = 0.25$ m, $h \in [0.25, 0.5]$ m, and $d \in [4, 7.5]$ m. The *mixed* terrains are constructed by successive uniform random selection of gap, wall, and step features. The parameter ranges are the same as for the individual environments.

State description: The state consists of a concatenation of a reduced character state and the terrain state. For the dog, a 5-dimensional reduced state is defined by

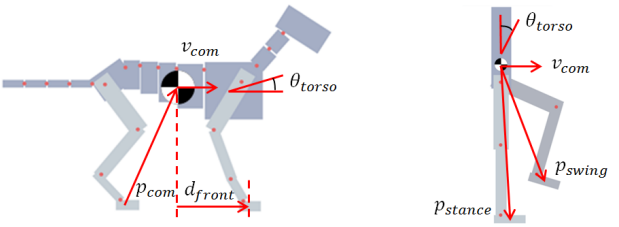

Figure 9: *State features for the dog and the biped.*

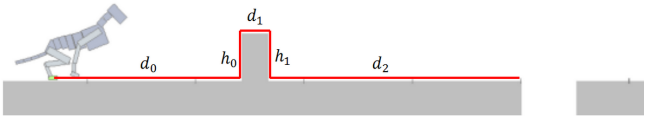

Figure 10: *Terrain features used in the state description.*

$(p_{\mathrm{com.x}}, p_{\mathrm{com.y}}, v_{\mathrm{com.x}}, d_{\mathrm{front}}, \theta_{\mathrm{torso}})$, as illustrated in Figure 9. The terrain state is defined by 5 features, $(d_0, h_0, d_1, h_1, d_2)$, as shown in Figure 10, where d_0 is measured relative to the position of the rear foot, which is always in a stance phase at the start of an action. The final combined state representation is thus 10-dimensional.

Similarly, the biped uses a 6-dimensional reduced character state description defined by $(p_{stance}, p_{swing}, v_{com.x}, \theta_{torso})$, as shown in Figure 9. The terrain state is identical to that used for the dog, with d_0 being measured relative to the stance foot. The final combined state is 11-dimensional.

Task rewards: The reward function for our terrain traversal tasks is given by a function $R(s, a, s')$, i.e., it is a function of the start state, end state, and chosen action, and it reflects the desirability of the state transition. For brevity, we drop the (s, a, s') dependencies in the descriptions that follow. We begin by defining $R_c(s, a, s')$ which contains terms that are common to the dog and biped reward functions:

$$R_c = \begin{cases} 0 & : \text{ character falls} \\ 0.5 + w_v E_v + w_e E_e + w_s E_s & : \text{ otherwise} \end{cases}$$

where E_v, E_e, and E_s correspond to rewards (or penalties) for velocity, effort, and stumbles, respectively, $w_v = 0.0002$, $w_e = 0.0004$, $w_s = 0.0025$,

$$E_v = \mathrm{clamp}(\frac{\bar{v}}{2}, 0, 1),$$

$$E_e = \exp(10^{-6} \sum_{j \in \mathrm{joints}} \int \tau_j^2 dt),$$

$$E_s = 1/(1 + 0.2n),$$

and n is the number of times the character stumbles during a given action.

The reward function for the dog for a given bound or jump is then defined by

$$R_{\mathrm{dog}} = \begin{cases} 0 & : \text{ dog falls} \\ R_c + w_d E_d & : \text{ otherwise} \end{cases},$$

where $w_d = 0.001$,

$$E_d = \mathrm{clamp}(2d_{\mathrm{edge}}, 0, 1),$$

and d_{edge} is the distance to the leading edge of the nearest upcoming obstacle in state s', equivalent to d_0..

The reward function for the biped for a given running step or leap is defined by

$$R_{\text{biped}} = \begin{cases} 0 & : \text{ biped falls} \\ R_c + w_p E_p + w_f E_f & : \text{ otherwise} \end{cases},$$

where $w_p = 0.001$, $w_f = 0.001$,

$$E_p = 1 - \text{clamp}(0.5(|\theta| - 0.4), 0, 1),$$

θ is the pitch of the torso in state s',

$$E_f = \begin{cases} \min(3.3h_{\text{clear}}, 1) & : \text{ jumping over a wall} \\ 1 & : \text{ otherwise} \end{cases},$$

and h_{clear} is the minimum clearance height of the feet above the obstacle.

A discount factor of $\gamma = 0.9$ is used to compute the expected cumulative rewards as modeled by the value function.

7 Results

We synthesize control policies for various terrains and characters. Policies are computed in five phases: (1) initial exploration + synthesis (ES) that collects 50k tuples (100k for the biped); (2) 65 iterations of ES, with 5k tuples/iteration; (3) distance metric optimization; (4) 130 iterations of ES, with 5k tuples/iteration; and (5) tuple-reduction. Approximately 1M total tuples are generated for each scenario, resulting in $\sim$ 500k winning tuples and $\sim$ 300k tuples after tuple-reduction. The full pipeline takes 25.6 hours for the dog and 4.8 hours for the biped. The biped simulations are faster because the biped has fewer links. The compute time for the dog and biped is (17.9 h, 1.4 h) for distance metric optimization, (2.5 h, 2.9 h) for value iteration, (4.0 h, 0.4 h) for state exploration, and (0.8 h, 0.1 h) for tuple reduction. These results are on an 8-core machine, with a mulithreaded implementation for all phases.

The two videos which accompany this paper provide the best illustration of the learned terrain traversal skills. Figures 11–13 show the learned skills of the dog for traversing terrains with *gaps*, *steps*, and *walls*. Separate policies are learned for each class of terrain. A policy is also learned for a *mixed* terrain class that randomly mixes gaps, steps, and walls, seen in Figure 14. Figures 15 and 16 similarly illustrate the skills learned for the biped for traversing terrains with *walls* and *gaps*.

Policy Approximation: In order to evaluate the importance of kNN function approximation with outlier removal, we compare its performance with a simple nearest-neighbor approach and also with a kNN interpolation that does not remove outliers. Table 4 compares the mean distance traveled for the three methods, as tested on policies for different classes of terrain for both the biped (top) and the dog (bottom). The kNN policy with outlier removal significantly outperforms the other interpolation methods. We speculate that the simple NN policy suffers from overfitting to a single policy action, while a failure to remove outliers results in unintended interpolation between actions that are significantly different, as previously described.

Global vs Local Exploration: We examine the effectiveness of using global exploration as well as local exploration during the ϵ-greedy exploration. We compare local-only with global+local exploration for learning policies for the *dog+mixed*, *dog+gaps*, and *biped+gaps* terrains. The results indicate no significant performance difference for the dog scenarios, but better performance for global+local for *biped+gaps*. We speculate that the utility of global exploration depends on the degree to which actions are compatible with a diverse range of states for a given application domain.

scenario	kNN + outlier removal	NN	kNN
dog + steps	704	275	52
dog + gaps	305	181	83
dog + walls	413	165	56
dog + mixed	237	154	55
biped + gaps	394	65	54
biped + walls	213	109	59

Table 4: *Impact of outlier removal and interpolation on policy performance. The performance is measured in terms of mean distance travelled, in metres, before a fall, as computed on 128 terrains*

feature	uniform	gaps	steps	walls	mixed
d_0	0.1	0.168	0.270	0.198	0.192
h_0	0.1	0.068	0.263	0.017	0.068
d_1	0.1	0.163	0.004	0.165	0.163
h_1	0.1	0.156	0.006	0.118	0.274
d_2	0.1	0.026	0.058	0.002	0.068
$v_{\text{com,x}}$	0.1	0.118	0.083	0.024	0.002
d_{front}	0.1	0.039	0.127	0.151	0.020
$p_{\text{com,x}}$	0.1	0.083	0.081	0.132	0.074
$p_{\text{com,y}}$	0.1	0.057	0.035	0.105	0.071
θ_{torso}	0.1	0.122	0.073	0.088	0.068
σ_λ^2	0.2	0.200	0.260	0.250	0.212

Table 5: *Distance metric weights computed for the dog scenarios.*

Distance Metric Optimization: The normalized weights that result from the distance metric optimization step in the learning pipeline are given in Table 5. The features they weight have already been standardized with respect to their mean and standard deviation. The first four features listed in the table are generally weighted the most, indicating the importance of anticipating upcoming terrain.

In Table 6 we compare the effect of running the learning pipeline as we have defined it, i.e., with an optimized distance metric, and with a default uniformly-weighted distance metric, i.e., skipping the distance metric optimization step. The results indicate the significant performance benefit of the optimized distance metric.

In order to further determine the sensitivity of policies with respect to the specific choice of distance metric, we evaluate the performance of the learned *dog + gaps* policy when applied using the distance metrics learned for each type of terrain, as well as with the uniform distance metric. In these tests, we only change the distance metric and do not rerun value iteration. The evaluation computes the average distance traveled before a fall for 128 terrains. The resulting performance measures are {305,279,256,141} m for { *gaps, mixed, walls, steps* }, respectively. The best results are thus ob-

scenario	uniform	optimized
dog + gaps	196.8	304.5
dog + mixed	162.2	236.5
biped + gaps	223.6	394.3

Table 6: *Performance impact of the distance metric, as measured in mean distance travelled (m).*

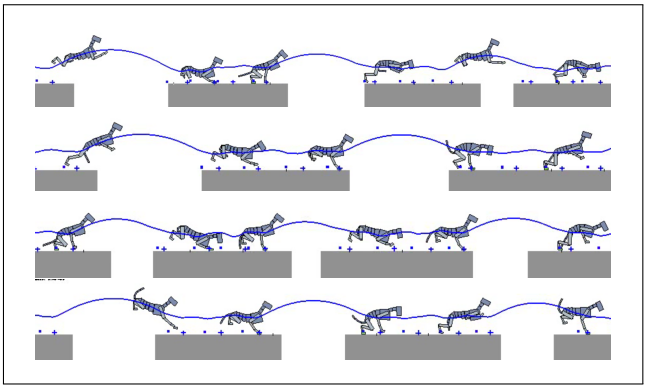

Figure 11: *Dog traversing "gaps" terrain.*

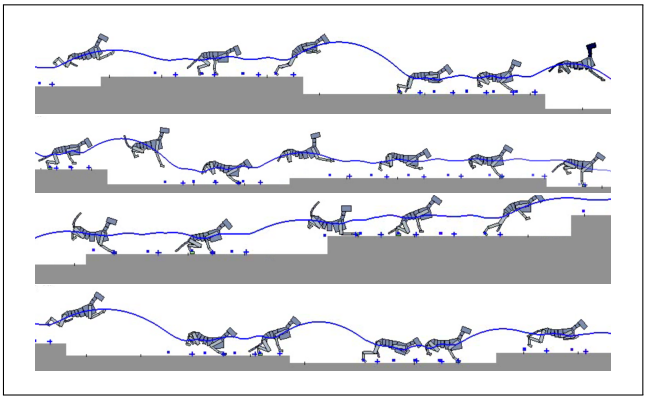

Figure 12: *Dog traversing "steps" terrain.*

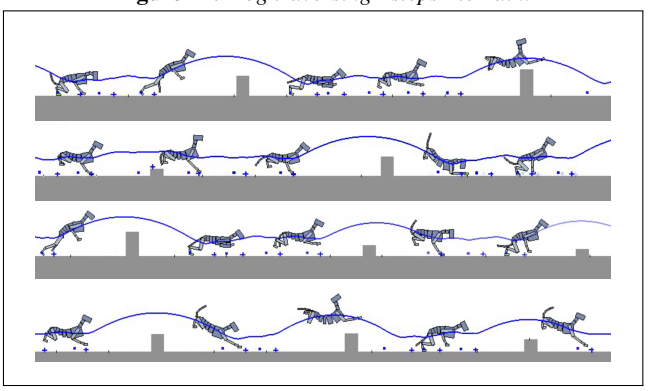

Figure 13: *Dog traversing "walls" terrain.*

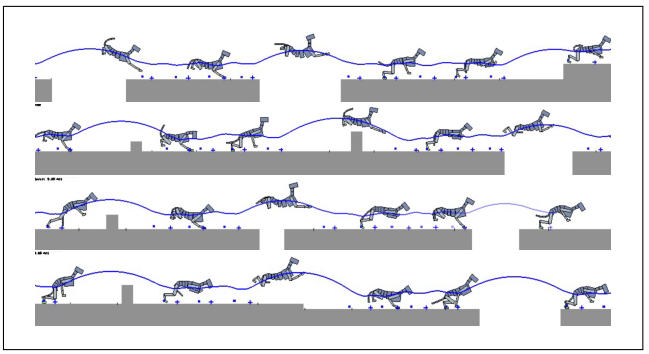

Figure 14: *Dog traversing "mixed" terrain.*

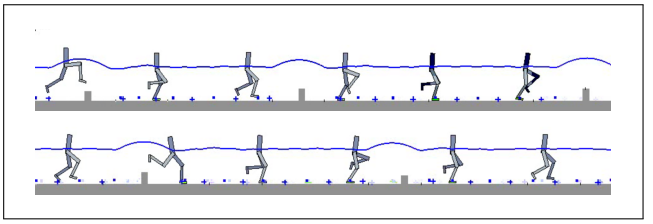

Figure 15: *Biped traversing "walls" terrain.*

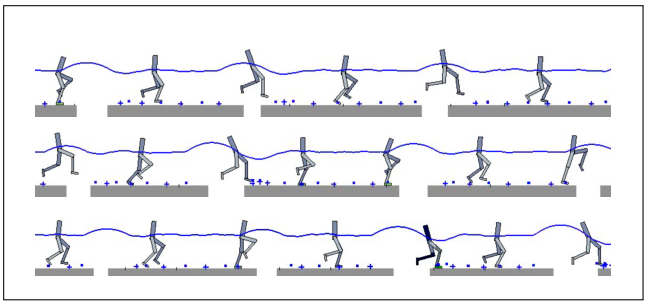

Figure 16: *Biped traversing "gaps" terrain.*

tained with the distance metric optimized for the gaps terrain, with the distance metric for the mixed terrain, which also contains gaps, being a close second. The steps distance metric is poorly suited for the gaps task. In order to test the importance of the body-state features, i.e., features 6–10 in Table 5, we compute a modified version of the *gaps* distance metric by assigning weights of zero to these features and then renormalizing the remaining weights for features 1–5, i.e., the terrain features. We again evaluate this distance metric for the *dog + gaps* policy, yielding an average distance of 83 m. This poor performance is indicative of the body state being critical to the final policy performance.

Discrete Action Sets vs Continuous Action Spaces: We also evaluate the benefits that are afforded by a continuous action space by comparing the policy performance to that of policies obtained using a discrete action space, as defined by the set of initial actions that we use to seed the learning process. For the discrete action policy, the exploration steps now draw uniformly from the set of discrete actions, and a voting scheme is used for policy approximation instead of kNN approximation.

We compare the discrete and continuous action policies for the dog on several challenging terrain scenarios. The results are summarized in Table 7 and are expressed in terms of the percentage of obstacles traversed that result in failures as evaluated during 128 terrain episodes. The *big gaps* and *big walls* each have their continuous policies initialized using the set of 8 initial seed actions, which also define the actions available to the discrete-action policy that we compute for comparison. The results show that the discrete policies fail at more than double the rate of continuous policies.

We also tested the ability of the pipeline to adapt jumps for new situations by training it in the tight gaps terrain, which has gaps of width $w \in [1, 3]$ m and spaced at a distance $d \in [2.5, 4]$ m. With shorter recovery distances, the character is forced to jump before it can return to more stable states. The results again illustrate that the failure rate for the discrete-actions policy is double that of the continuous policy. Additionally, we tested the learning pipeline when seeded with fewer initial actions. We developed a policy for the dog in the regular gaps terrain using only 3 initial actions. The performance of the final continuous policy is equivalent to that of the

name	description	continuous	discrete
big gaps	5.75 m gaps	30%	64%
big walls	0.75 m walls	20%	52%
tight gaps	gap spacing $d \in [2.5, 4]$ m	5.4%	11.2%

Table 7: *Performance comparison of continuous actions and discrete action sets for the dog. The performance is given as the percentage of attempted object traversals that fail.*

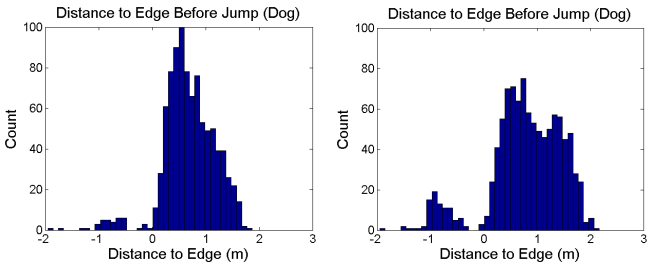

Figure 17: *Histograms of the distance of the last foot contact from the leading edge of a gap for the dog, as sampled for the continuous policy (left) and the discrete policy (right).*

continuous policy initialized with 8 actions: 303 m before a fall, vs 304 m before a fall. This indicates the ability of continuous action exploration to do well even with very few initial actions.

We further investigate the benefits of continuous actions in two other ways. Figure 17 gives histograms of the distance of the last foot contact from the leading edge of a gap for the dog. When effort is included in the objective function, we expect these distances to be small because it allows for smaller, lower-effort jumps. Small distances are thus better, although they also entail a risk of falling into the gap, and negative distances indicate a jump into the gap. As seen in Figure 17 (left), the histogram for the continuous policy peaks at small positive distances, while the results for the discrete policy are on average much further from the edge, and there are also more negative-distance samples, i.e., failures.

Figure 18 gives a scatterplot of the jump effort as a function of the gap size. The continuous policy shows a progressive increase in effort as a function of gap size, while the discrete policy has horizontal bands that correspond to the discrete actions. As a result, the continuous policy is generally able to outperform the discrete policy in terms of effort expended.

8 Conclusions

We have presented the successful application of reinforcement learning (RL) to the control of dynamic jumping and leaping motions for articulated figures across challenging terrains. We demonstrate tuple-based non-parametric approximations, value iteration based on positive temporal differences, and epsilon-greedy exploration. Taken together, these can be an effective strategy for motion control problems that are characterized by high-dimensional-and-continuous state spaces and action spaces. Importantly, an embodied state representation is used which contains both character state features and environment features, and a distance metric is then learned for this composite state representation. The use of correlated epsilon-greedy exploration allows for new actions and new states to be explored in a progressive fashion. An important aspect for our problem domain is the use of outlier-culling in the policy approximation, which avoids the undesired averaging of conflicting actions.

Our method has a number of limitations that remain to be addressed. The majority of the compute time is spent on distance metric optimization, although we expect that this could be reduced in a variety of ways. We currently make use of non-trivial prior domain knowledge when designing the action parameterization. An open question is the degree to which similar results can be achieved with more naive action parameterizations. We have made initial attempts at extending the method to handle more generalized obstacles. For example, it would be exciting if the policy could discover when it is best to leap *onto* a "wall" and when it is best to leap *over* the wall. We have not yet been successful with achieving well-performing policies for such problems.

We believe that the current method can likely achieve 3D motions that are similar in nature to the current planar motions, i.e., where the terrain features and primary motion are predominantly within the sagittal plane. We expect that many highly-dynamic, momentum-based 3D motions would fit this characterization. However, motions on more arbitrary 3D terrains that require sharp turns or that have restrictive foot placements are likely to require additional sophistication. Our non-parametric approach may be unable to generate sufficient examples to cover the large space of possibilities. We expect that such scenarios would benefit from online motion planning, something that is not yet part of our current method.

We wish to develop more aggressive strategies for culling experience tuples that do not make a significant contribution to the solution and to explore more sophisticated non-parametric representations. Convergence to an optimal policy or even local optima for our CACLA-inspired algorithm is not yet guaranteed, although we have not found convergence to be an issue. We wish to investigate online motion planning as an alternative strategy for dealing with more complex terrains,

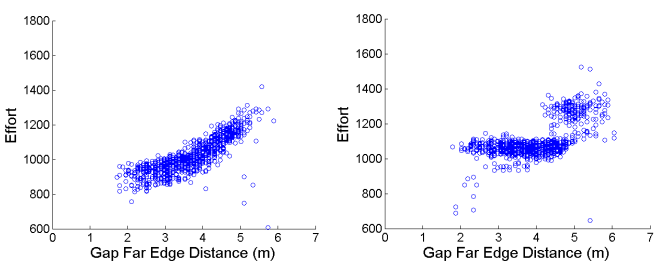

Figure 18: *Effort vs gap size for the continuous policy (left) and the discrete policy (right) for the dog gaps scenario.*

References

BOX2D, 2015. Box2d: A 2d physics engine for games, Jan. http://box2d.org.

BUSONIU, L., BABUSKA, R., DE SCHUTTER, B., AND ERNST, D. 2010. *Reinforcement learning and dynamic programming using function approximators*. CRC press.

COROS, S., BEAUDOIN, P., YIN, K. K., AND VAN DE PANN, M. 2008. Synthesis of constrained walking skills. *ACM Trans. Graph. 27*, 5, Article 113.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2009. Robust task-based control policies for physics-based characters. *ACM Transctions on Graphics 28*, 5, Article 170.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2010. Generalized biped walking control. *ACM Transctions on Graphics 29*, 4, Article 130.

COROS, S., KARPATHY, A., JONES, B., REVERET, L., AND VAN DE PANNE, M. 2011. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics 30*, 4, Article 59.

DE LASA, M., MORDATCH, I., AND HERTZMANN, A. 2010. Feature-based locomotion controllers. *ACM Transactions on Graphics (TOG) 29*, 4, 131.

ENGEL, Y., MANNOR, S., AND MEIR, R. 2005. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, ACM, 201–208.

FONTENEAU, R., MURPHY, S. A., WEHENKEL, L., AND ERNST, D. 2013. Batch mode reinforcement learning based on the synthesis of artificial trajectories. *Annals of operations research 208*, 1, 383–416.

GEIJTENBEEK, T., AND PRONOST, N. 2012. Interactive character animation using simulated physics: A state-of-the-art review. In *Computer Graphics Forum*, vol. 31, 2492–2515.

HA, S., AND LIU, C. K. 2015. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Transactions on Graphics (TOG)*. to appear.

HA, S., YE, Y., AND LIU, C. K. 2012. Falling and Landing Motion Control for Character Animation. *ACM Transactions on Graphics 31*, 6 (Nov.), 1.

HANSEN, N. 2006. The cma evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, 75–102.

JAIN, S., YE, Y., AND LIU, C. K. 2009. Optimization-based interactive motion synthesis. *ACM Transactions on Graphics (TOG) 28*, 1, 10.

KWON, T., AND HODGINS, J. 2010. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *Proc. of Symposium on Computer Animation*, 129–138.

LANGE, S., GABEL, T., AND RIEDMILLER, M. 2012. Batch reinforcement learning. In *Reinforcement Learning*. Springer, 45–73.

LEE, J., AND LEE, K. H. 2006. Precomputing avatar behavior from human motion data. *Graphical Models 68*, 2, 158–174.

LEE, Y., LEE, S. J., AND POPOVIĆ, Z. 2009. Compact character controllers. *ACM Transctions on Graphics 28*, 5, Article 169.

LEE, Y., WAMPLER, K., BERNSTEIN, G., POPOVIĆ, J., AND POPOVIĆ, Z. 2010. Motion fields for interactive character locomotion. *ACM Transctions on Graphics 29*, 6, Article 138.

LEE, Y., KIM, S., AND LEE, J. 2010. Data-driven biped control. *ACM Transctions on Graphics 29*, 4, Article 129.

LEVINE, S., AND KOLTUN, V. 2014. Learning complex neural network policies with trajectory optimization. In *Proc. ICML 2014*, 829–837.

LEVINE, S., WANG, J. M., HARAUX, A., POPOVIĆ, Z., AND KOLTUN, V. 2012. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics 31*, 4, 28.

LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph. 31*, 6, 154.

MACCHIETTO, A., ZORDAN, V., AND SHELTON, C. R. 2009. Momentum control for balance. In *ACM Transactions on Graphics (TOG)*, vol. 28, ACM, 80.

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

MORDATCH, I., DE LASA, M., AND HERTZMANN, A. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph. 29*, 4, Article 71.

MUJA, M., AND LOWE, D. G. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, 331–340.

ORMONEIT, D., AND SEN, Ś. 2002. Kernel-based reinforcement learning. *Machine learning 49*, 2-3, 161–178.

RAIBERT, M. H., AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. In *ACM SIGGRAPH Computer Graphics*, vol. 25, ACM, 349–358.

ROSS, S., GORDON, G., AND BAGNELL, A. 2011. A reduction of imitation learning and structured prediction to noregret online learning. *Journal of Machine Learning Research 15*, 627–635.

STEWART, A. J., AND CREMER, J. F. 1992. Beyond keyframing: an algorithmic approach to animation. In *Graphics Interface*, 273–281.

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

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

VAN HASSELT, H., AND WIERING, M. A. 2007. Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, IEEE, 272–279.

VAN HASSELT, H. 2012. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*. Springer, 207–251.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2009. Optimizing walking controllers. *ACM Transctions on Graphics 28*, 5, Article 168.

WEI, X., MIN, J., AND CHAI, J. 2011. Physically valid statistical models for human motion generation. *ACM Transctions on Graphics 30*, 3, Article 19.

YE, Y., AND LIU, C. K. 2010. Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph. 29*, 4, Article 74.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Transctions on Graphics 26*, 3, Article 105.

YIN, K., COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2008. Continuation methods for adapting simulated skills. *ACM Transctions on Graphics 27*, 3, Article 81.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *Proc. of Symposium on Computer Animation*, 89–96.