Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning

Xue Bin Peng, Glen Berseth, Michiel van de Panne*
University of British Columbia

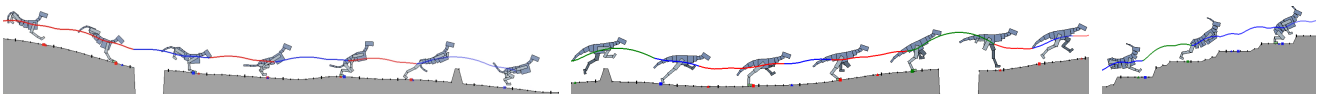


Figure 1: *Terrain traversal using a learned actor-critic ensemble. The color-coding of the center-of-mass trajectory indicates the choice of actor used for each leap.*

Abstract

Reinforcement learning offers a promising methodology for developing skills for simulated characters, but typically requires working with sparse hand-crafted features. Building on recent progress in deep reinforcement learning (DeepRL), we introduce a mixture of actor-critic experts (MACE) approach that learns terrain-adaptive dynamic locomotion skills using high-dimensional state and terrain descriptions as input, and parameterized leaps or steps as output actions. MACE learns more quickly than a single actor-critic approach and results in actor-critic experts that exhibit specialization. Additional elements of our solution that contribute towards efficient learning include Boltzmann exploration and the use of initial actor biases to encourage specialization. Results are demonstrated for multiple planar characters and terrain classes.

Keywords: physics-based characters, reinforcement learning

Concepts: •**Computing methodologies** → **Animation; Physical simulation;**

1 Introduction

Humans and animals move with grace and agility through their environment. In animation, their movement is most often created with the help of a skilled animator or motion capture data. The use of reinforcement learning (RL) together with physics-based simulations offers the enticing prospect of developing classes of motion skills from first principles. This requires viewing the problem through the lens of a sequential decision problem involving states, actions, rewards, and a control policy. Given the current situation of the character, as captured by the state, the control policy decides on the best action to take, and this then results in a subsequent state, as well as a reward that reflects the desirability of the observed state transition. The goal of the control policy is to maximize the sum of expected future rewards, i.e., any immediate rewards as well as all future rewards.

*xbpeng|gberseth|van@cs.ubc.ca

SIGGRAPH '16 Technical Paper,, July 24 - 28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925881

ACM Reference Format
Peng, X., Berseth, G., van de Panne, M. 2016. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. ACM Trans. Graph. 35, 4, Article 81 (July 2016), 12 pages. DOI = 10.1145/2897824.2925881 http://doi.acm.org/10.1145/2897824.2925881.

In practice, a number of challenges need to be overcome when applying the RL framework to problems with continuous and high-dimensional states and actions, as required by movement skills. A control policy needs to select the best actions for the distribution of states that will be encountered, but this distribution is often not known in advance. Similarly, the distribution of actions that will prove to be useful for these states is also seldom known in advance. Furthermore, the state-and-action distributions are not static in nature; as changes are made to the control policy, new states may be visited, and, conversely, the best possible policy may change as new actions are introduced. It is furthermore not obvious how to best represent the state of a character and its environment. Using large descriptors allows for very general and complete descriptions, but such high-dimensional descriptors define large state spaces that pose a challenge for many RL methods. Using sparse descriptors makes the learning more manageable, but requires domain knowledge to design an informative-and-compact feature set that may nevertheless be missing important information.

Contributions: In this paper we use deep neural networks in combination with reinforcement learning (DeepRL) to address the above challenges. This allows for the design of control policies that operate directly on high-dimensional character state descriptions (83D) and an environment state that consists of a height-field image of the upcoming terrain (200D). We provide a parameterized action space (29D) that allows the control policy to operate at the level of bounds, leaps, and steps. We introduce a novel *mixture of actor-critic experts (MACE) architecture* to enable accelerated learning. MACE develops n individual control policies and their associated value functions, which each then specialize in particular regimes of the overall motion. During final policy execution, the policy associated with the highest value function is executed, in a fashion analogous to Q-learning with discrete actions. We show the benefits of *Boltzmann exploration* and various algorithmic features for our problem domain. We demonstrate improvements in motion quality and terrain abilities over previous work.

2 Related Work

Physics-based Character Animation:
Significant progress has been made in recent years in developing methods to create motion from first principles, i.e., control and physics, as a means of character animation [Geijtenbeek and Pronost 2012]. Many methods focus mainly on controlling physics-based locomotion over flat terrain. Some methods assume no access to the equations of motion and rely on domain knowledge to develop simplified models that can be used in the design of controllers that are commonly developed around a phase-based state machine, e.g., [Hodgins et al. 1995; Laszlo et al. 1996; Yin et al. 2007; Sok et al. 2007; Coros et al. 2010; Lee et al. 2010b]. Inverse-

dynamics based methods also provide highly effective solutions, with the short and long-term goals being encoded into the objectives of a quadratic program that is solved at every time-step, e.g., [da Silva et al. 2008; Muico et al. 2009; Mordatch et al. 2010; Ye and Liu 2010]. Many methods use some form of model-free policy search, wherein a controller is first designed and then has a number of its free parameters optimized using episodic evaluations, e.g., [Yin et al. 2008; Wang et al. 2009; Coros et al. 2011; Liu et al. 2012; Tan et al. 2014]. Our work uses the control structures found in much of the prior work in this area to design the action parameterization, but then goes on to learn controllers that can move with agility across different classes of terrain, a capability that has been difficult to achieve with prior methods.

Reinforcement learning as guided by value or action-value functions have also been used for motion synthesis. In particular, this type of RL has been highly successful for making decisions about which motion clip to play next in order to achieve a given objective [Lee and Lee 2006; Treuille et al. 2007; Lee et al. 2009; Lee et al. 2010a; Levine et al. 2012]. Work that applies value-function-based RL to the difficult problem of controlling the movement of physics-based characters has been more limited [Coros et al. 2009; Peng et al. 2015] and has relied on the manual selection of a small set of important features that represent the state of the character and its environment.

Deep Neural Networks (DNNs):
Control policies based on DNNs have been learned to control motions such as swimming [Grzeszczuk et al. 1998], as aided by a differentiable neural network approximation of the physics. The recent successes of deep learning have also seen a resurgence of its use for learning control policies, These can be broadly characterized into three categories, which we now elaborate on.

Direct policy approximation: DNNs can be used to directly approximate a control policy, $a = \pi(s)$ from example data points (s_i, a_i) as generated by some other control process. For example, trajectory optimization can be used to compute families of optimal control solutions from various initial states, and each trajectory then yields a large number of data points suitable for supervised learning. A naive application of these ideas will often fail because when the approximated policy is deployed, the system state will nevertheless easily drift into regions of state space for which no data has been collected. This problem is rooted in the fact that a control policy and the state-distribution that it encounters are tightly coupled. Recent methods have made progress on this issue by proposing iterative techniques. Optimal trajectories are generated in close proximity to the trajectories resulting from the current policy; the current policy is then updated with the new data, and the iteration repeats. These *guided policy search* methods have been applied to produce motions for robust bipedal walking and swimming gaits for planar models [Levine and Koltun 2014; Levine and Abbeel 2014] and a growing number of challenging robotics applications. Relatedly, methods that leverage contact-invariant trajectory optimization have also demonstrated many capabilities, including planar swimmers, planar walkers, and 3D arm reaching [Mordatch and Todorov 2014], and, more recently, simulated 3D swimming and flying, as well as 3D bipeds and quadrupeds capable of skilled interactive stepping behaviors [Mordatch et al. 2015]. In this most recent work, the simulation is carried out as a state optimization, which allows for large timesteps, albeit with the caveat that dynamics is a soft constraint and thus may result in some residual root forces. A contribution of our work is to develop model-free methods that are complementary to the model-based trajectory optimization methods described above.

Deep Q-Learning: For decision problems with discrete action spaces, a DNN can be used to approximate a set of value functions, one for each action, thereby learning a complete state-action value (Q) function. A notable achievement of this approach has been the ability to learn to play a large suite of Atari games at a human-level of skill, using only raw screen images and the score as input [Mnih et al. 2015]. Many additional improvements have since been proposed, including prioritized experience replay [Schaul et al. 2015], double-Q learning [Van Hasselt et al. 2015], better exploration strategies [Stadie et al. 2015], and accelerated learning using distributed computation [Nair et al. 2015]. However, it is not obvious how to directly extend these methods to control problems with continuous action spaces.

Policy gradient methods: In the case of continuous actions learned in the absence of an oracle, DNNs can be used to model both the Q-function, $Q(s, a)$, and the policy $\pi(s)$. This leads to a single composite network, $Q(s, \pi(s))$, that allows for the back-propagation of value-gradients back through to the control policy, and therefore provides a mechanism for policy improvement. Since the original method for using deterministic policy gradients [Silver et al. 2014], several variations have been proposed with a growing portfolio of demonstrated capabilities. This includes a method for stochastic policies that can span a range of model-free and model-based methods [Heess et al. 2015], as demonstrated on examples that include a monoped, a planar biped walker, and an 8-link planar cheetah model. Recently, further improvements have been proposed to allow end-to-end learning of image-to-control-torque policies for a wide selection of physical systems [Lillicrap et al. 2015]. While promising, the resulting capabilities and motion quality as applied to locomoting articulated figures still fall well short of what is needed for animation applications. Another recent work proposes the use of policy-gradient DNN-RL with parameterized controllers for simulated robot soccer [Hausknecht and Stone 2015], and theoretically-grounded algorithms have been proposed to ensure monotonic policy improvement [Schulman et al. 2015]. Our work develops an alternative learning method to policy-gradient methods and demonstrates its capability to generate agile terrain-adaptive motion for planar articulated figures.

Mixure-of-Experts and Ensemble methods: The idea of modular selection and identification for control has been proposed in many variations. Well-known work in sensorimotor control proposes the use of a responsibility predictor that divides experiences among several contexts [Haruno et al. 2001]. Similar concepts can be found in the use of skill libraries indexed based on sensory information [Pastor et al. 2012], Gaussian mixture models for multi-optima policy search for episodic tasks [Calinon et al. 2013], and the use of random forests for model-based control [Hester and Stone 2013]. Ensemble methods for RL problems with discrete actions have been investigated in some detail [Wiering and Van Hasselt 2008]. Adaptive mixtures of local experts [Jacobs et al. 1991] allow for specialization by allocating learning examples to a particular expert among an available set of experts according to a local gating function, which is also learned so as to maximize performance. This has also been shown to work well in the context of reinforcement learning [Doya et al. 2002; Uchibe and Doya 2004]. More recently, there has been strong interest in developing deep RL architectures for multi-task learning, where the tasks are known in advance [Parisotto et al. 2015; Rusu et al. 2015], with a goal of achieving policy compression. In the context of physics-based character animation, a number of papers propose to use selections or combinations of controllers as the basis for developing more complex locomotion skills [Faloutsos et al. 2001; Coros et al. 2008; da Silva et al. 2009; Muico et al. 2011].

Our work: We propose a deep reinforcement learning method based on learning Q-functions and a policy, $\pi(s)$, for continuous action spaces as modeled on the CACLA RL algorithm [Van Hasselt and Wiering 2007; Van Hasselt 2012]. In particular, we show

the effectiveness of using a *mixture of actor-critic experts (MACE)*, as constructed from multiple actor-critic pairs that each specialize in particular aspects of the motion. Unlike prior work on dynamic terrain traversal using reinforcement learning [Peng et al. 2015], our method can work directly with high-dimensional character and terrain state descriptions without requiring the feature engineering often needed by non-parametric methods. Our results also improve on the motion quality and expand upon the types of terrains that can be navigated with agility.

3 Overview

An overview of the system is shown in Figure 2, which illustrates three nested loops that each correspond to a different time scale. In the following description, we review its operation for our dog control policies, with other control policies being similar.

The inner-most loop models the low-level control and physics-based simulation process. At each time-step δt, individual joint torques are computed by low-level control structures, such as PD-controllers and Jacobian transpose forces (see §4). These low-level control structures are organized into a small number of motion phases using a finite state machine. The motion of the character during the time step is then simulated by a physics engine.

The middle loop operates at the time scale of locomotion cycles, i.e., leaps for the dog. Touch-down of the hind-leg marks the beginning of a motion cycle, and at this moment the control policy, $a = \pi(s)$, chooses the action that will define the subsequent cycle. The state, s, is defined by C, a set of 83 numbers describing the character state, and T, a set of 200 numbers that provides a one-dimensional heightfield "image" of the upcoming terrain. The output action, a, assigns specific values to a set of 29 parameters of the FSM controller, which then governs the evolution of the motion during the next leap.

The control policy is defined by a small set of actor-critic pairs, whose outputs taken together represent the outputs of the learned deep network (see Figure 8). Each actor represents an individual control policy; they each model their own actions, $A_\mu(s)$, as a function of the current state, s. The critics, $Q_\mu(s)$, each estimate the quality of the action of their corresponding actor in the given situation, as given by the Q-value that they produce. This is a scalar that defines the objective function, i.e., the expected value of the cumulative sum of (discounted) future rewards. The functions $A_\mu(s)$ and $Q_\mu(s)$ are modeled using a single deep neural network that has multiple corresponding outputs, with most network layers being shared. At run-time, the critics are queried at the start of each locomotion cycle in order to select the actor that is best suited for the current state, according to the highest estimated Q-value. The output action of the corresponding actor is then used to drive the current locomotion cycle.

Learning requires exploration of "off-policy" behaviors. This is implemented in two parts. First, an actor can be selected probabilistically, instead of deterministically choosing the max-Q actor. This is done using a *softmax*-based selection, which probabilistically selects an actor, with higher probabilities being assigned to actor-critic pairs with larger Q-values. Second, Gaussian noise can be added to the output of an actor with a probability ϵ_t, as enabled by an exploration choice of $\lambda = 1$.

For learning purposes, each locomotion cycle is summarized in terms of an experience tuple $\tau = (s, a, r, s', \mu, \lambda)$, where the parameters specify the starting state, action, reward, next state, index of the active actor, and a flag indicating the application of exploration noise. The tuples are captured in a replay memory that stores the most recent 50k tuples and is divided into a critic buffer and

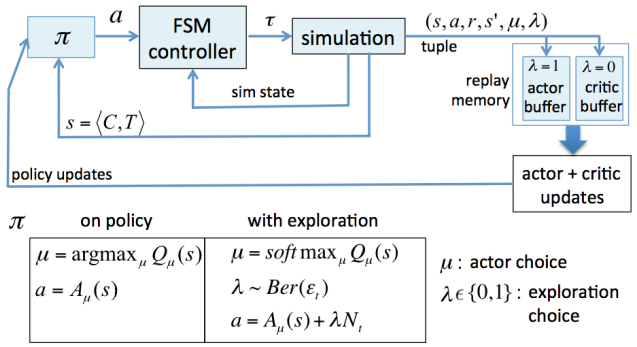


Figure 2: *System Overview*

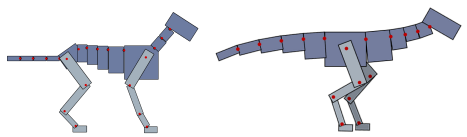


Figure 3: *21-link planar dog (left), and 19-link raptor (right).*

an actor buffer. Experiences are collected in batches of 32 tuples, with the motion being restarted as needed, i.e., if the character falls. Tuples that result from added exploration noise are stored in the actor buffer, while the remaining tuples are stored in the critic buffer, which are later used to update the actors and critics respectively. Our use of actor buffers and critic buffers in a MACE-style architecture is new, to the best of our knowledge, although the actor buffer is inspired by recent work on prioritized experience replay [Schaul et al. 2015].

The outer loop defines the learning process. After the collection of a new minibatch of tuples, a learning iteration is invoked. This involves sampling minibatches of tuples from the replay memory, which are used to improve the actor-critic experts. The actors are updated according to a positive-temporal difference strategy, modeled after CACLA [Van Hasselt 2012], while the critics are updated using the standard temporal difference updates, regardless of the sign of their temporal differences. For more complex terrains, learning requires on the order of 300k iterations.

4 Characters and Terrains

Our planar dog model is a reconstruction of that used in previous work [Peng et al. 2015], although it is smaller, standing approximately 0.5 m tall at the shoulders, as opposed to 0.75 m. It is composed of 21 links and has a mass of 33.7 kg. The pelvis is designated as the root link and each link is connected to its parent link with a revolute joint, yielding a total of 20 internal degrees of freedom and a further 3 degrees of freedom defined by the position and orientation of the root in the world. The raptor is composed of 19 links, with a total mass of 33 kg, and a head-to-tail body length of 1.5 m. The motion of the characters are driven by the application of internal joint torques and is simulated using the Bullet physics engine [Bullet 2015] at 600 Hz, with friction set to 0.81.

4.1 Controllers

Similar to much prior work in physics-based character animation, the motion is driven using joint torques and is guided by a finite state machine. Figure 4 shows the four phase-structure of the con-

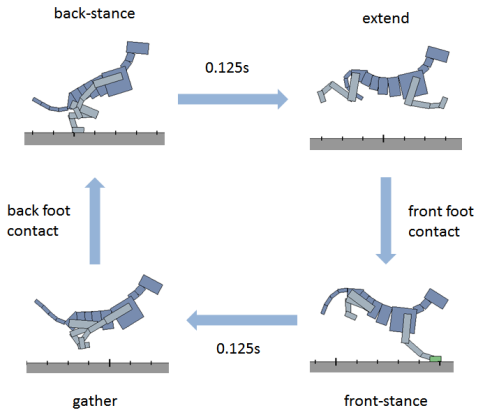


Figure 4: *Dog controller motion phases.*

troller. In each motion phase, the applied torques can be decomposed into three components,

$$\tau = \tau_{\text{spd}} + \tau_g + \tau_{\text{vf}}$$

where τ_{spd} are torques computed from joint-specific stable (semi-implicit) proportional-derivative (SPD) controllers [Tan et al. 2011], τ_g provides gravity compensation for all links, as referred back to the root link, and τ_{vf} implements virtual leg forces for the front and hind legs when they are in contact with the ground, as described in detail in [Peng et al. 2015]. The stable PD controllers are integrated into Bullet with the help of a Featherstone dynamics formulation [Featherstone 2014]. For our system, conventional PD-controllers with explicit Euler integration require a time-step $\delta t = 0.0005s$ to remain stable, while SPD remains stable for a time-step of $\delta t = 0.0017s$, yielding a two-fold speedup once the setup computations are taken into account.

The character's propulsion is principally achieved by exerting forces on the ground with its end effectors, represented by the front and back feet. Virtual force controllers are used to compute the joint torques τ_e needed to exert a desired force f_e on a particular end effector e.

$$\tau_e = \delta_e J_e^T f_e$$

where δ_e is the contact indicator variable for the end effector, and J_e is the end effector Jacobian. The final control forces for the virtual force controllers are the sum of the control forces for the front and back feet.

$$\tau_{vf} = \tau_f + \tau_b$$

4.2 Terrain classes

We evaluate the learning method on multiple classes of terrain obstacles that include gaps, steps, walls, and slopes. It is possible to make the obstacles arbitrarily difficult and thus we use environments that are challenging while remaining viable. All of the terrains are represented by 1D height-fields, and generated randomly by drawing uniformly from predefined ranges of values for the parameters that characterize each type of obstacle. In the flat terrains, gaps are spaced between 4 to 7m apart, with widths ranging from 0.5 to 2m, and a fixed gap depth of 2m. Steps are spaced 5 to 7m apart, with changes in height ranging from 0.1 to 0.4m. Walls are spaced 6 to 8m apart, with heights ranging between 0.25 to 0.5m, and a fixed width of 0.2m. Slopes are generated by varying the change in slope of the terrain at each vertex following a momentum

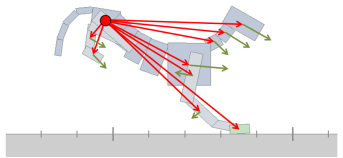


Figure 5: *The character features consist of the displacements of the centers of mass of all links relative to the root (red) and their linear velocities (green).*

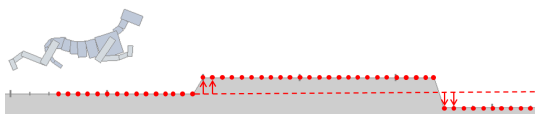


Figure 6: *Terrain features consist of height samples of the terrain in front of the character, evenly spaced 5cm apart. All heights are expressed relative to the height of the ground immediately under the root of the character.*

model. The height y_i of vertex i is computed according to

$$y_i = y_{i-1} + \triangle x s_i$$

$$s_i = s_{i-1} + \triangle s_i$$

$$\triangle s_i = \text{sign}(U(-1,1) - \frac{s_{i-1}}{s_{max}}) \times U(0, \triangle s_{max})$$

where $s_{max} = 0.5$ and $\triangle s_{max} = 0.05$, $\triangle x = 0.1$m, and vertices are ordered such that $x_{i-1} < x_i$. When slopes are combined with the various obstacles, the obstacles are adjusted to be smaller than those in the flat terrains.

5 Policy Representation

A policy is a mapping between a state space S and an action space A, i.e., $\pi(s) : S \mapsto A$. For our framework, S is a continuous space that describes the state of the character as well as the configuration of the upcoming terrain. The action space A is represented by a 29D continuous space where each action specifies a set of parameters to the FSM. The following sections provide further details about the policy representation.

5.1 State

A state s consists of features describing the configuration of the character and the upcoming terrain. The state of the character is represented by its pose q and velocity $\dot{q}$, where q records the positions of the center of mass of each link with respect to the root and $\dot{q}$ records the center of mass velocity of each link. The terrain features, T, consist of a 1D array of samples from the terrain height-field, beginning at the position of the root and spanning 10 m ahead. All heights are expressed relative to the height of the terrain immediately below the root of the character. The samples are spaced 5 cm apart, for a total of 200 height samples. Combined, the final state representation is 283-dimensional. Figure 5 and 6 illustrate the character and terrain features.

5.2 Actions

A total of 29 controller parameters serve to define the available policy actions. These include specifications of the target spine curvature as well as the target joint angles for the shoulder, elbow, hip,

knee, hock, and hind-foot, for each of the four motion phases defined by the controller FSM. Additionally, the x and y components of the hind-leg and front-leg virtual forces, as applied in phases 1 and 3 of the controller, are also part of the parameter set. Phases 2 and 4 apply the same forces as phases 1 and 3, respectively, if the relevant leg is still in contact with the ground. Lastly, the velocity feedback gain for the swing hip (and shoulder) provides one last action parameter.

Prior to learning the policy, a small set of initial actions are created which are used to seed the learning process. The set of actions consists of 4 runs and 4 leaps. All actions are synthesized using a derivative-free optimization process, CMA [Hansen 2006]. Two runs are produced that travel at approximately 4 m/s and 2 m/s, respectively. These two runs are then interpolated to produce 4 runs of varying speeds. Given a sequence of successive fast-run cycles, a single cycle of that fast-run is then optimized for distance traveled, yielding a 2.5 m leap that can then be executed from the fast run. The leap action is then interpolated with the fast run to generate 4 parametrized leaps that travel different distances.

5.3 Reward

In reinforcement learning the reward function, $r(s, a, s')$, is used as a training signal to encourage or discourage behaviors in the context of a desired task. The reward provides a scalar value reflecting the desirability of a particular state transition that is observed by performing action a starting in the initial state s and resulting in a successor state s'. Figure 7 is an example of a sequence of state transitions for terrain traversal. For the terrain traversal task, the reward is provided by

$$r(s, a, s') = \begin{cases} 0, & \text{character falls during the cycle} \\ e^{-\omega(v^* - v)^2}, & \text{otherwise} \end{cases}$$

where a fall is defined as any link of the character's trunk making contact with the ground for an extended period of time, v is the average horizontal velocity of the center of mass during a cycle, $v^* = 4m/s$ is the desired velocity, and $\omega = 0.5$ is the weight for the velocity error. This simple reward is therefore designed to encourage the character to travel forward at a consistent speed without falling. If the character falls during a cycle, it is reset to a default state and the terrain is regenerated randomly.

The goal of learning is to find a control policy that maximizes the expected value of the cumulative reward, R. Here, R can be expressed as the time-discounted sum of all transition rewards, r_i, from the current action up to a horizon T, where T may be infinite, i.e.,

$$R(s_0) = r_0 + \gamma r_1 + ... + \gamma^T r_T$$

$r_i = r(s_i, a_i, s'_i)$ and $\gamma = 0.9$ is a discount factor. The sequence of states and actions are determined by the policy and the dynamics of the system. $\gamma \in [0, 1)$ ensures that the cumulative reward is bounded, and captures the intuition that events occurring in the distant future are likely to be of less consequence than those occuring in the more immediate future. The summation can be rewritten recursively as

$$R(s_0) = r_0 + \gamma \sum_{i=1}^{T} \gamma^{i-1} r_i = r_0 + \gamma R(s_1)$$

This recursive property provides the foundations for temporal difference learning, which lies at the heart of many RL algorithms.

5.4 Policy Representation

To represent the policy, we use a convolutional neural network, with weights θ, following the structure illustrated in Figure 8. The network is queried once at the start of each locomotion cycle. The overall structure of the convolutional network is inspired by the recent work of Minh et al. [2015]. For a query state $s = (q, \dot{q}, T)$, the network first processes the terrain features T by passing it through 16 8×1 convolution filters. The resulting feature maps are then convolved with 32 4×1 filters, followed by another layer of 32 4×1 filters. A stride of 1 is used for all convolutional layers. The output of the final convolutional layer is processed by 64 fully connected units, and the resulting features are then concatenated with the character features q and $\dot{q}$, as inspired by [Levine et al. 2015]. The combined features are processed by a fully connected layer composed of 256 units. The network then branches into critic and actor subnetworks. The critic sub-network predicts the Q-values for each actor, while each actor subnetwork proposes an action for the given state. All subnetworks follow a similar structure with a fully connected layer of 128 units followed by a linear output layer. The size of the output layers vary depending on the subnetwork, ranging from 3 output units for the critics to 29 units for each actor. The combined network has approximately 570k parameters. Rectified linear units are used for all layers, except for the output layers.

During final runtime use, i.e., when learning has completed, the actor associated with the highest predicted Q-value is selected and its proposed action is applied for the given state.

$$\mu^* = \arg\max_{\mu} Q_\mu(s)$$

$$\pi(s) = A_{\mu^*}(s)$$

The inputs are standardized before being used by the network. The mean and standard deviation for this standardization are determined using data collected from an initial random-action policy. The outputs are also followed by the inverse of a similar transformation in order to allow the network to learn standardized outputs. We apply the following transformation:

$$A_\mu(s) = \Sigma \bar{A}_\mu(s) + \beta_\mu$$

where $\bar{A}_\mu$ is the output of each actor subnetwork, $\Sigma = diag(\sigma_0, \sigma_1, ...)$, $\{\sigma_i\}$ are pre-specified scales for each action parameter, and β_μ is an actor bias. $\{\sigma_i\}$ are selected such that the range of values for each output of $\bar{A}_\mu$ remains approximately within [-1, 1]. This transformation helps to prevent excessively small or large gradients during back-propagation, improving stability during learning. The choice of different biases for each actor helps to encourage specialization of the actors for different situations, a point which we revisit later. For the dog policy, we select a fast run, slow run, and large jump as biases for the three actors.

6 Learning

Algorithm 1 illustrates the overall learning process. θ represents the weights of the composite actor-critic network, and $Q_\mu(s|\theta)$ is

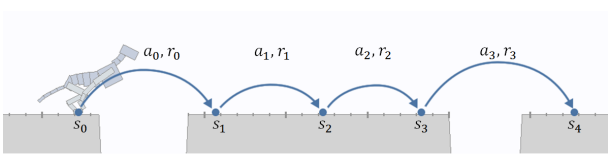


Figure 7: *Each state transition can be recorded as a tuple $\tau_i = (s_i, a_i, r_i, s'_i)$. s_i is the initial state, a_i is the action taken, s'_i is the resulting state, and r_i is the reward received during the ith cycle.*

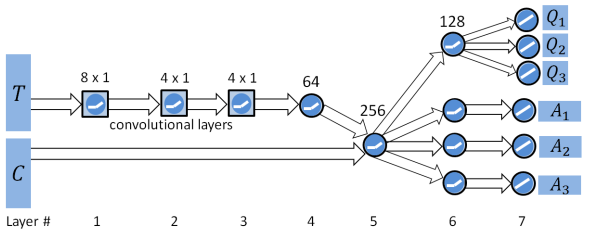


Figure 8: *Schematic illustration of the MACE convolutional neural network. T and C are the input terrain and character features. Each A_μ represents the proposed action of actor μ, and Q_μ is the critic's predicted reward when activating the corresponding actor.*

the Q-value predicted by the critic for the result of activating actor A_μ in state s, where $A_\mu(s|\theta)$ is the action proposed by the actor for s. Since an action is decided by first selecting an actor followed by querying the chosen actor for its proposed action, exploration in MACE can be decomposed into **critic exploration** and **actor exploration**. Critic exploration allows for the selection of an actor other than the "best" actor as predicted by the Q-values. For this we use Boltzmann exploration, which assigns a selection probability p_μ to each actor based on its predicted Q-value:

$$p_\mu(s) = \frac{e^{Q_\mu(s|\theta)/T_t}}{\sum_j e^{Q_{\mu_j}(s|\theta)/T_t}},$$

where T_t is a temperature parameter. Actors with higher predicted values are more likely to be selected, and the bias in favor of actors with higher Q-values can be adjusted via T_t. Actor exploration results in changes to the output of the selected actor, in the form of Gaussian noise that is added to the proposed action. This generates a new action from the continuous action space according to:

$$a = A_\mu(s) + \mathcal{N}(0, \Sigma^2),$$

where Σ are pre-specified scales for each action parameter. Actor exploration is enabled via a Bernoulli selector variable, $\lambda \sim \text{Ber}(\epsilon_t)$: $\text{Ber}(\epsilon_t) = 1$ with probability ϵ_t, and $\text{Ber}(\epsilon_t) = 0$ otherwise.

Once recorded, experience tuples are stored into separate replay buffers, for use during updates. Tuples collected during actor exploration are stored in an actor buffer D_a, while all other tuples are stored in a critic buffer D_c. This separation allows the actors to be updated using only the off-policy tuples in D_a, and the critics to be updated using only tuples without exploration noise in D_c During a critic update, a minibatch of $n = 32$ tuples $\{\tau_i\}$ are sampled from D_c and used to perform a Bellman backup,

$$y_i = r_i + \gamma \max_\mu Q_\mu(s_i'|\theta)$$

$$\theta \leftarrow \theta + \alpha \left(\frac{1}{n} \sum_i (y_i - Q_{\mu_i}(s_i|\theta)) \frac{\partial Q_{\mu_i}(s_i|\theta)}{\partial \theta} \right)$$

During an actor update, a minibatch of tuples $\{\tau_j\}$ are sampled from D_a and a CACLA-style positive-temporal difference update is applied to each tuple's respective actor,

$$\delta_j = y_j - \max_\mu Q_\mu(s_j|\theta)$$

$$\text{if } \delta_j > 0 : \theta \leftarrow \theta + \alpha \left(\frac{1}{n} (a_j - A_{\mu_j}(s_j|\theta)) \frac{\partial A_{\mu_j}(s_j|\theta))}{\partial \theta} \right)$$

Algorithm 1 MACE

$\theta \leftarrow$ random weights
Initialize D_c and D_a with tuples from a random policy

while not done **do**
  for $step = 1, ..., m$ **do**
    $s \leftarrow$ character and terrain initial state
    $\mu \leftarrow$ select each actor with probability $p_{\mu_i} = \frac{\exp(Q_{\mu_i}(s|\theta)/T_t)}{\sum_j (\exp(Q_{\mu_j}(s|\theta)/T_t))}$
    $\lambda \leftarrow Ber(\epsilon_t)$
    $a \leftarrow A_\mu(s|\theta) + \lambda N_t$
    Apply a and simulate forward 1 cycle
    $s' \leftarrow$ character and terrain terminal state
    $r \leftarrow$ reward
    $\tau \leftarrow (s, a, r, s', \mu, \lambda)$
    if $\lambda = 1$ **then**
      Store τ in D_a
    else
      Store τ in D_c
    end if
  end for

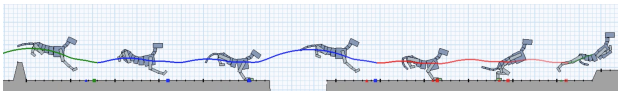
  Update critic:
  Sample minibatch of n tuples $\{\tau_i = (s_i, a_i, r_i, s_i', \mu_i, \lambda_i)\}$ from D_c
  $y_i \leftarrow r_i + \gamma \max_\mu Q_\mu(s_i'|\theta)$ for each τ_i
  $\theta \leftarrow \theta + \alpha(\frac{1}{n} \sum_i (y_i - Q_{\mu_i}(s_i|\theta)) \frac{\partial Q_{\mu_i}(s_i|\theta)}{\partial \theta})$

  Update actors:
  Sample minibatch of n tuples $\{\tau_j = (s_j, a_j, r_j, s_j', \mu_j, \lambda_j)\}$ from D_a
  for each τ_j **do**
    $y_j \leftarrow \max_\mu Q_\mu(s_j|\theta)$
    $y_j' \leftarrow r_j + \gamma \max_\mu Q_\mu(s_j'|\theta)$
    if $y_j' > y_j$ **then**
      $\theta \leftarrow \theta + \alpha(\frac{1}{n}(a_j - A_{\mu_j}(s_j|\theta)) \frac{\partial A_{\mu_j}(s_j|\theta))}{\partial \theta})$
    end if
  end for
end while

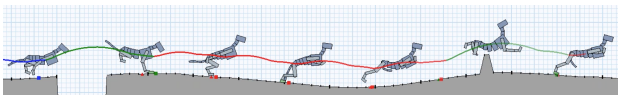
Target network: Similarly to [Mnih et al. 2015], we used a separate target network when computing the target values y_i during updates. The target network is fixed for 500 iterations, after which it is updated by copying the most up-to-date weights θ, and then held fixed again for another 500 iterations.

Hyperparameter settings: $m = 32$ steps are simulated before each update. Updates are performed using stochastic gradient descent with momentum, with a learning rate, $\alpha = 0.001$, a weight decay of 0.0005 for regularization, and momentum set to 0.9. ϵ_t is initialized to 0.9 and linearly annealed to 0.2 after 50k iterations. Similarly, the temperature T_t used in Boltzmann exploration is initialized to 20 and linearly annealed to 0.025 over 50k iterations.

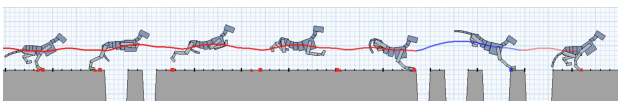
Initialization: The actor and critic buffers are initialized with 50k tuples from a random policy that selects an action uniformly from the initial action set for each cycle. Each of the initial actions are manually associated with a subset of the available actors using the actor bias. When recording an initial experience tuple, μ will be randomly assigned to be one of the actors in its respective set.



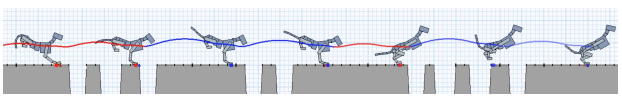
(a) `mixed` terrain



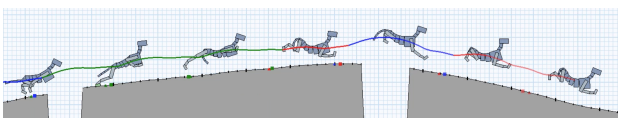
(b) `slopes-mixed` terrain



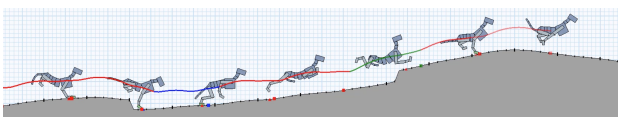
(c) `narrow-gaps` terrain



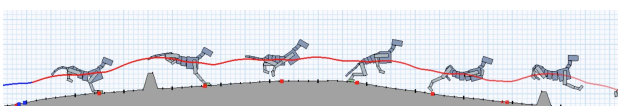
(d) `tight-gaps` terrain



(e) `slopes-gaps` terrain



(f) `slopes-steps` terrain



(g) `slopes-walls` terrain

Figure 9: *Dog control policies.*

7 Results

The motions resulting from the learned policies are best seen in the supplemental video. The majority of the results we present are on policies for the dog, as learned for the 7 different classes of terrain shown in Figure 9. By default, each policy uses three actor-critic pairs. The final policies are the result of 300k iterations of training, collecting about 10 million tuples, and requiring approximately 20h of compute time on a 16-core cluster, using a multithreaded C++ implementation. All networks are built and trained using Caffe [Jia et al. 2014]. Source code is available at https://github.com/xbpeng/DeepTerrainRL. The learning time remains dominated by the cost of simulating the motion of the character rather than the neural network updates. Because of this, we did not pursue the use of GPU-accelerated training. Once the control policies have been learned, all results run faster than real time. Exploration is turned off during the evaluation of the control policies. Separate policies are learned for each class of terrain. The development of a single policy that would be capable of all these terrain classes is left as future work.

In order to evaluate attributes of the learning algorithm, we use the mean distance before a fall as our performance metric, as measured across 250 evaluations. We do not use the Q-function estimates computed by the policies for evaluation as these can over time provide inflated estimates of the cumulative rewards that constitute the true objective.

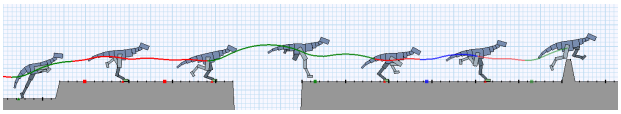
Scenario	Performance (m)
dog + `mixed`: MACE(3)	2094
dog + `mixed`: Q	194
dog + `mixed`: CACLA	1095
dog + `slopes-mixed`: MACE(3)	1364
dog + `slopes-mixed`: Q	110
dog + `slopes-mixed`: CACLA	739
dog + `narrow-gaps`: MACE(3)	176
dog + `narrow-gaps`: Q	74
dog + `narrow-gaps`: CACLA	38
dog + `tight-gaps`: MACE(3)	44
dog + `slopes-gaps`: MACE(3)	1916
dog + `slopes-steps`: MACE(3)	3782
dog + `slopes-walls`: MACE(3)	4312
goat + `variable-steps`: MACE(3)	1004
raptor + `mixed`: MACE(3)	1111
raptor + `slopes-mixed`: MACE(3)	562
raptor + `narrow-gaps`: MACE(3)	145

Table 1: *Performance of the final policies.*

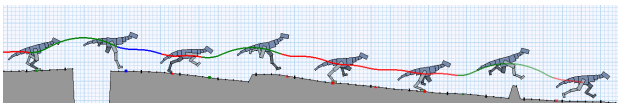
We compare the final performance of using a mixture of three actor-critic experts, i.e., MACE(3), to two alternatives. First, we compare to Double Q-learning using a set of DNN approximators, $Q_i(s)$, for each of the 8 initial actions [Van Hasselt et al. 2015]. This is motivated by the impressive success of DNNs in learning control policies for discrete action spaces [Mnih et al. 2015]. We use a similar DNN architecture to the network shown in Figure 8, except for the exclusion of the actor subnetworks. Second, we compare against the CACLA algorithm [Van Hasselt 2012]. In our case, CACLA is identical to MACE(1), except without the use of a critic buffer, which means that CACLA uses all experience tuples to update the critic, while MACE(1) only uses the tuples generated without applied exploration noise. In practice, we often find the performance of CACLA and MACE(1) to be similar.

Table 1 gives the final **performance numbers** for all the control policies, including the MACE/Q/CACLA comparisons. The final mean performance may not always tell the full story, and thus histograms of the performance distributions are provided in the supplemental material, i.e., Figures 16–19, and we also compare the learning curves, as we shall discuss shortly. MACE(3) significantly outperforms Q-learning and CACLA for all three terrain classes used for comparison. In two out of three terrain classes, CACLA outperforms Q-learning with discrete actions. This may reflect the importance of being able to learn new actions. As measured by their final performance, all three approaches rank the terrains in the same order in terms of difficulty: `narrow-gaps` (hardest), `slopes-mixed`, `mixed` (easiest). The `tight-gaps` terrain proved to be the most difficult for MACE(3); it consists of the same types of gaps as `narrow-gaps`, but with half the recovery distance between sequences of gaps.

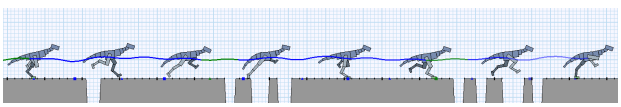
In addition to the dog, we apply MACE(3) to learn control policies for **other characters and terrain types**, as shown in Figure 10. The raptor model uses a 4-states-per-step finite state machine controller, with fixed 0.0825 s state transitions and a final state transition occurring when the swing foot strikes the ground. The control policy is invoked at the start of each step to make a decision with regard to the FSM parameters to apply in the next step. It uses a set of 28 control parameters, similar to those of the dog. We also



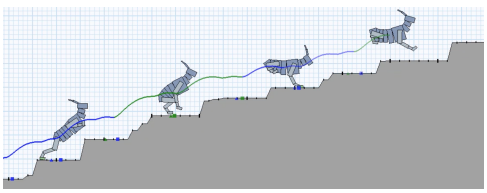
(a) Raptor `mixed` terrain



(b) Raptor `slopes-mixed` terrain



(c) Raptor `narrow-gaps` terrain



(d) Goat on `variable-steps` terrain

Figure 10: *Other Control Policies*

explored goat locomotion by training the dog to climb never-ending sequences of steep steps that have variable widths and heights. A set of 5 initial actions are provided, corresponding to jumps of heights varying from 0.2 m to 0.75 m, and 3 of which are used as the initial actor biases. Though the character uses the same underlying model as the dog, it is rendered as a goat in the figures and video as homage to the inspiration for this scenario.

In Figure 11, we provide **learning curve comparisons** for different architectures and algorithm features. As before, we evaluate performance by measuring the mean distance before a fall across 100 randomly generated terrains. Figure 11(a) compares MACE(3) with discrete-action-set Q-learning and CACLA, as measured for the dog on mixed terrain. The Q-learning plateaus after 50k iterations, while CACLA continues to improve with further learning iterations, but at a slower rate than MACE(3).

Figure 11(b) shows the **effects of disabling features** of MACE that are enabled by default. Boltzmann exploration has a significant impact on learning. This is likely because it helps to encourage specialization by selecting actors with high predicted Q-values more frequently, while also enabling exploration of multiple actors in cases where they share similar predicted values, thus helping to better disambiguate between the utilities of the different actors. The actor buffer has a large impact, as it allows learning to focus on exploratory actions that yielded an improvement. The initial use of actor bias also proves to be significant, which we attribute to the breaking of initial symmetry between the actor-critic pairs. Lastly, the critic buffer, which enables the critics to learn exclusively from the actions of the deterministic actor policies without exploration noise, does not show a significant benefit for this particular example. However, we found the critic buffer to yield improved learning in earlier experiments, and we further keep this feature because of the more principled learning that it enables.

Figure 11(c) shows the impact of the **number of actor-critic pairs**, comparing MACE(1), MACE(2), MACE(3), MACE(6) for the dog on `mixed` terrain. MACE(2) and MACE(3) yield the best learning performance, while MACE(6) results in a drop in learning performance, and MACE(1) is the worst. A larger number of actor-critic pairs allows for increased specialization, but results in fewer learning tuples for each actor-critic pair, given that our current MACE learning method only allows for a single actor-critic pair to learn from a particular tuple.

Scenario	Perf. (m)
`mixed` policy in `slopes-mixed`	80
`slopes-gaps` policy in `slopes-mixed`	35
`slopes-mixed` policy in `slopes-gaps`	1545

Table 2: *Performance of applying policies to unfamiliar terrains.*

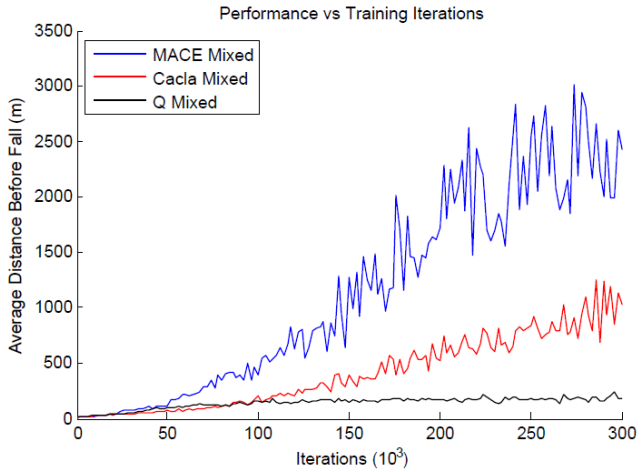
Learning good control policies requires **learning new actions** that yield improved performance. The MACE architecture supports actor-critic specialization by having multiple experts that can specialize in different motions, as well as taking advantage of unique biases for each actor to encourage diversity in their specializations. Figure 12 illustrates the space of policy actions early and late in the learning process. This visualization is created using t-SNE (t-distributed Stochastic Neighbor Embedding) [van der Maaten and Hinton 2008], where a single embedding is constructed using all the action samples collected at various iterations in the training process. Samples from each iteration are then rendered separately. The numbers embedded in the plots correspond to the set of 8 initial actions. The actions begin nearby the initial actions, then evolve over time as demanded by the task, while remaining specialized. The evolution of the actions during learning is best seen in the supplementary videos.

To encourage **actor specialization**, the actor-specific initialization bias helps to break symmetry early on in the learning process. Without this initial bias, the benefit of multiple actor-critic experts is diminished. Figure 13 illustrates that actor specialization is much less evident when all actors receive the same initial bias, i.e., the fast run for the dog.

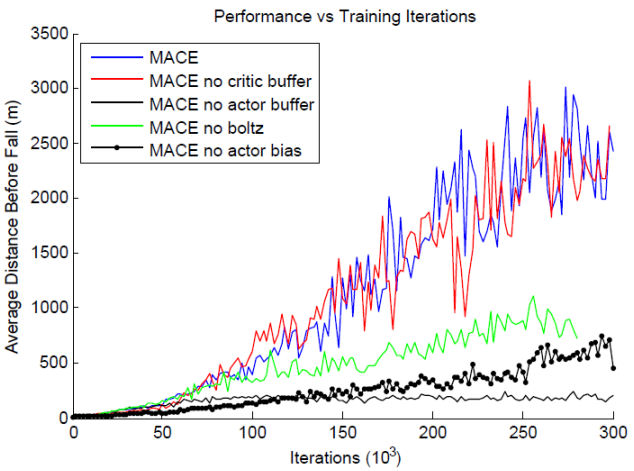
We test the **generalization capability** of the MACE(3) policy for the dog in the `slopes-mixed` terrain, which can be parameterized according to a scale parameter, ψ, that acts a multiplier for the size of all the gaps, steps, and walls. Here, $\psi > 1$ implies more difficult terrains, while $\psi < 1$ implies easier terrains. Figure 14 shows that the performance degrades gracefully as the terrain difficulty is increased. To test the performance of the policies when applied to unfamiliar terrain, i.e., terrains not encountered during training, we apply the policy trained in `mixed` to `slopes-mixed`, `slopes-gaps` to `slopes-mixed`, and `slopes-mixed` to `slopes-gaps`. Table 1 summarizes the results of each scenario. The policies perform poorly when encountering unfamiliar obstacles, such as slopes for the `mixed` policy and walls for the `slopes-gaps` policy. The `slopes-mixed` policy performs well in `slopes-gaps`, since it has been previously exposed to gaps in `slopes-mixed`, but nonetheless does not reach a similar level of performance as the policy trained specifically for `slopes-gaps`.

8 Discussion

The use of a **predefined action parameterization** helps to provide a significant degree of action abstraction as compared to other recent deep-RL methods that attempt to learn control policies that are directly based on joint torques, e.g., [Levine and Abbeel 2014; Levine and Koltun 2014; Mordatch and Todorov 2014; Mordatch et al. 2015; Heess et al. 2015; Lillicrap et al. 2015]. Instead of fo-



(a) MACE(3) vs CACLA and Q-learning on mixed terrain

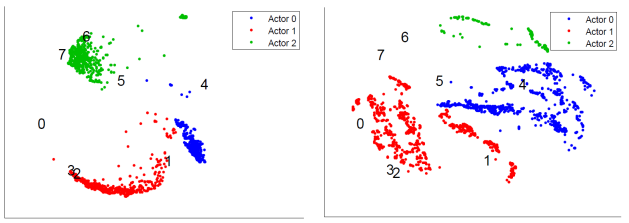


(b) MACE(3) with features disabled.

(c) MACE(n) for various number of actors.

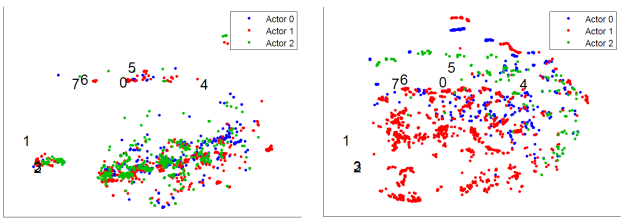
Figure 11: *Comparisons of learning performance.*

cusing on the considerable complexities and challenges of *tabula rasa* learning we show that deep-RL enables new capabilities for physics-based character animation, as demonstrated by agile dynamic locomotion across a multitude of terrain types. In future



(a) 10k iterations (b) 300k iterations

Figure 12: *Action space evolution for using MACE(3) with initial actor bias.*



(a) 10k iterations (b) 300k iterations

Figure 13: *Action space evolution for using MACE(3) without initial actor bias.*

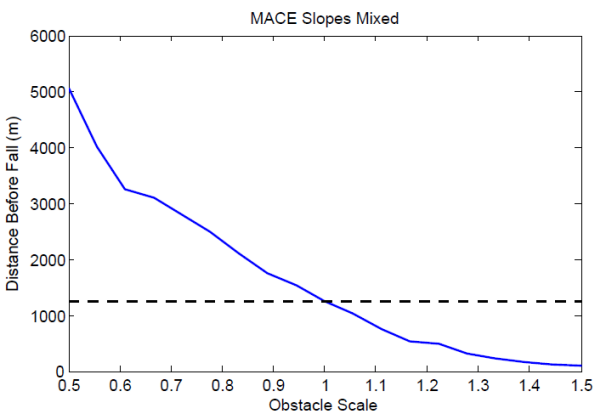


Figure 14: *Policy generalization to easier and more-difficult terrains.*

work, it will be interesting to explore the best ways of learning action abstractions and of determining the benefits, if any, of working with actuation that allows for control of stiffness, as allowed by PD-controllers or muscles.

Our overall learning approach is quite different from methods that interleave **trajectory optimization** (to generate reference data) and neural network regression for supervised learning of the control policy [Levine and Abbeel 2014; Levine and Koltun 2014; Mordatch and Todorov 2014; Mordatch et al. 2015]. The key role of trajectory optimization makes these methods strongly model-based, with the caveat that the models themselves can possibly be learned. In contrast, our approach does not need an oracle that can provide reference solutions. Another important difference is that much of our network is devoted to processing the high-dimensional terrain description that comprises a majority of our high-D state description.

Recent methods have also demonstrated the use of more direct **policy-gradient methods** with deep neural network architectures. This involves chaining a state-action value function approximator in sequence with a control policy approximator, which then allows for backpropagation of the value function gradients back to the control policy parameters, e.g., [Silver et al. 2014; Lillicrap et al. 2015]. Recent work applies this approach with predefined parameterized action abstractions [Hausknecht and Stone 2015]. We leave comparisons to these methods as important future work. To our knowledge, these methods have not yet been shown to be capable of highly dynamic terrain-adaptive motions. Our work shares many of the same challenges of these methods, such as making stationary-distribution assumptions which are then violated in practice. We do not yet demonstrated the ability to work directly with input images to represent character state features, as shown by others [Lillicrap et al. 2015].

We provide a qualitative **comparison to previous work** using non-parametric methods for similar terrain locomotion problems [Peng et al. 2015] in the supplemental video for this paper. We have not yet performed a quantitative comparison; the performance metrics are not directly comparable given the use of simulated dogs of different sizes as well as physics engines that exhibit differences in their modeling of contact friction, and further minor differences in the control parameterization. Our simulated dog has a shorter stride duration than the previous work, as might be expected from its smaller size. However, it travels at a similar absolute speed and therefore covers almost double the number of body lengths per unit time. Our work is distinct in its ability to learn policies from high-D state descriptors, without needing to rely on handcrafted character-state or terrain-state features. We would expect to see worse performance from the non-parametric method for terrains such as mixed-slopes and narrow gaps because of the difficulty of designing meaningful compact state descriptors for such terrains. We further note that only a single actor-critic is employed in [Peng et al. 2015], thereby making it most similar to CACLA and MACE(1) which exhibit significantly slower learning in our tests. Another salient difference is that on-policy updates to the critic in MACE can result in decreases to the state-value function for a given state, as should be the case, whereas this was not the case in the non-parametric approach because of its inherent optimism in retaining the best performing tuples.

We have encountered **difficult terrains** for which learning does not succeed, such as those that have small scale roughness, i.e., bumps 20-40 cm in width that are added to the other terrain features. With extensive training of 500k iterations, the dog is capable of performing robust navigation across the `tight-gaps` terrain, thereby in some sense "aiming for" the best intermediate landing location. However, we have not yet seen that a generalized version of this capability can be achieved, i.e., one that can find a suitable sequence of foot-holds if one exists. Challenging terrains may need to learn new actions and therefore demand a carefully staged learning process. A common failure mode is that of introducing overly-challenging terrains which therefore always cause early failures and a commensurate lack of learning progress.

There remain many **architecture hyperparameters** that we set based on limited experimentation, including the number of layers, the number of units per layer, regularization parameters, and learning batch size. The space of possible network architectures is large and we have explored this in only a minimal way. The magnitude of exploration for each action parameter is currently manually specified; we wish to develop automated solutions for this. Also of note is that all the actors and critics for any given controller currently share most of their network structure, branching only near the last layers of the network. We would like to explore the benefit, if any, of allowing individual critic and actor networks, thereby allowing additional freedom to utilize more representational power for the relevant aspects of their specialization. We also note that a given actor in the mixture may become irrelevant if its expected performance, as modeled by its critic, is never better than that of its peers for all encountered states. This occurs for MACE(3) when applied to the goat on `variable-steps`, where the distribution of actor usages is (57, 0, 43), with all figures expressed as percentages. Other example usage patterns are (43,48,9) for the dog on `mixed`, (23,66,11) for dog on `slopes-mixed`, and (17,73,10) for dog on `narrow-gaps`. One possible remedy to explore in future work would be to reinitialize an obsolete actor-critic pair with a copy of one of its more successful peers.

We wish to develop principled methods for **integrating multiple controllers** that are each trained for a specific class of terrain. This would allow for successful divide-and-conquer development of control strategies. Recent work has tackled this problem in domains involving discrete actions [Parisotto et al. 2015]. One obvious approach is to use another mixture model, wherein each policy is queried for its expected Q-value, which is in turn modeled as the best Q-value of its individual actors. In effect, each policy would perform its own analysis of the terrain for the suitability of its actions. However, this ignores the fact that the Q-values also depend on the expected distributions of the upcoming character states and terrain, which remain specific to the given class of terrain. Another problem is the lack of a model for the uncertainty of Q-value estimates. Nevertheless, this approach may yield reasonable results in many situations that do not demand extensive terrain-specific anticipation. The addition of models to predict the state would allow for more explicit prediction and planning based on the available set of controllers. We believe that the tradeoff between implicit planning, as embodied by the actor-critic network, and explicit planning, as becomes possible with learned forward models, will be a fruitful area for further reseach.

We have not yet demonstrated **3D terrain adaptive locomotion**. We expect that the major challenge for 3D will arise in the case of terrain structure that requires identifying specific feasible foothold sequences. The capacity limits of a MACE(n) policy remain unknown.

Control policies for difficult terrains may need to be learned in a progressive fashion via some form of **curriculum learning**, especially for scenarios where the initial random policy performs so poorly that no meaningful directions for improvement can be found. Self-paced learning is also a promising direction, where the terrain difficulty is increased once a desired level of competence is achieved with the current terrain difficulty. It may be possible to design the terrain generator to work in concert with the learning process by synthesizing terrains with a bias towards situations that are known to be problematic. This would allow for "purposeful practice" and for learning responses to rare events. Other paths towards more data-efficient learning include the ability to transfer aspects of learned solutions between classes of terrain, developing an explicit reduced-dimensionality action space, and learning models of the dynamics, e.g., [Assael et al. 2015]. It would also be interesting to explore the coevolution of the character and its control policies.

It is not yet clear how to best enable **control of the motion style**. The parameterization of the action space, the initial bootstrap actions, and the reward function all provide some influence over the final motion styles of the control policy. Available reference motions could be used to help develop the initial actions or used to help design style rewards.

9 Conclusions

We have presented a novel deep reinforcement learning architecture, based on CACLA-style learning and a mixture of actor-critic experts. We identify a number of other architectural features that lead to improved performance, including initial actor biases, separate replay buffers for actor and critic updates, and Boltzmann exploration. The architecture enables the development of control policies that can work directly with high-dimensional state descriptions for highly-dynamic terrain adaptive locomotion. This avoids the need to engineer compact hand-crafted feature descriptors and allows policies to be learned for classes of terrain for which it may not be easy to develop compact feature descriptors. We believe that the state of the art in physics-based character control will see rapid and significant advances as enabled by deep RL methods.

10 Acknowledgments

We thank KangKang Yin for her significant help in providing computational resources for this project, the anonymous reviewers for their helpful feedback. and NSERC for funding this research via a Discovery Grant (RGPIN-2015-04843).

References

ASSAEL, J.-A. M., WAHLSTRÖM, N., SCHÖN, T. B., AND DEISENROTH, M. P. 2015. Data-efficient learning of feedback policies from image pixels using deep dynamical models. *arXiv preprint arXiv:1510.02173*.

BULLET, 2015. Bullet physics library, Dec. http://bulletphysics.org.

CALINON, S., KORMUSHEV, P., AND CALDWELL, D. G. 2013. Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning. *Robotics and Autonomous Systems 61*, 4, 369–379.

COROS, S., BEAUDOIN, P., YIN, K. K., AND VAN DE PANNE, 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.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Interactive simulation of stylized human locomotion. *ACM Trans. Graph. 27*, 3, Article 82.

DA SILVA, M., DURAND, F., AND POPOVIĆ, J. 2009. Linear bellman combination for control of character animation. *ACM Trans. Graph. 28*, 3, Article 82.

DOYA, K., SAMEJIMA, K., KATAGIRI, K.-I., AND KAWATO, M. 2002. Multiple model-based reinforcement learning. *Neural computation 14*, 6, 1347–1369.

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

FEATHERSTONE, R. 2014. *Rigid body dynamics algorithms*. Springer.

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

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proc. ACM SIGGRAPH*, ACM, 9–20.

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

HARUNO, M., WOLPERT, D. H., AND KAWATO, M. 2001. Mosaic model for sensorimotor learning and control. *Neural computation 13*, 10, 2201–2220.

HAUSKNECHT, M., AND STONE, P. 2015. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*.

HEESS, N., WAYNE, G., SILVER, D., LILLICRAP, T., EREZ, T., AND TASSA, Y. 2015. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2926–2934.

HESTER, T., AND STONE, P. 2013. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine Learning 90*, 3, 385–429.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of SIGGRAPH 1995*, 71–78.

JACOBS, R. A., JORDAN, M. I., NOWLAN, S. J., AND HINTON, G. E. 1991. Adaptive mixtures of local experts. *Neural computation 3*, 1, 79–87.

JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, ACM, New York, NY, USA, MM '14, 675–678.

LASZLO, J., 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.

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 ABBEEL, P. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems 27*. 1071–1079.

LEVINE, S., AND KOLTUN, V. 2014. Learning complex neural network policies with trajectory optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 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 (TOG) 31*, 4, 28.

LEVINE, S., FINN, C., DARRELL, T., AND ABBEEL, P. 2015. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*.

LILLICRAP, T. P., HUNT, J. J., PRITZEL, A., HEESS, N., EREZ, T., TASSA, Y., SILVER, D., AND WIERSTRA, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

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.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. 2015. Human-level control through deep reinforcement learning. *Nature 518*, 7540, 529–533.

MORDATCH, I., AND TODOROV, E. 2014. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*.

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

MORDATCH, I., LOWREY, K., ANDREW, G., POPOVIC, Z., AND TODOROV, E. V. 2015. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, 3114–3122.

MUICO, U., LEE, Y., POPOVIĆ, J., AND POPOVIĆ, Z. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph. 28*, 3, Article 81.

MUICO, U., POPOVIĆ, J., AND POPOVIĆ, Z. 2011. Composite control of physically simulated characters. *ACM Trans. Graph. 30*, 3, Article 16.

NAIR, A., SRINIVASAN, P., BLACKWELL, S., ALCICEK, C., FEARON, R., DE MARIA, A., PANNEERSHELVAM, V., SULEYMAN, M., BEATTIE, C., PETERSEN, S., ET AL. 2015. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*.

PARISOTTO, E., BA, J. L., AND SALAKHUTDINOV, R. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

PASTOR, P., KALAKRISHNAN, M., RIGHETTI, L., AND SCHAAL, S. 2012. Towards associative skill memories. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, IEEE, 309–315.

PENG, X. B., BERSETH, G., AND VAN DE PANNE, M. 2015. Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics 34*, 4.

RUSU, A. A., COLMENAREJO, S. G., GULCEHRE, C., DESJARDINS, G., KIRKPATRICK, J., PASCANU, R., MNIH, V., KAVUKCUOGLU, K., AND HADSELL, R. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295*.

SCHAUL, T., QUAN, J., ANTONOGLOU, I., AND SILVER, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

SCHULMAN, J., LEVINE, S., MORITZ, P., JORDAN, M. I., AND ABBEEL, P. 2015. Trust region policy optimization. *CoRR abs/1502.05477*.

SILVER, D., LEVER, G., HEESS, N., DEGRIS, T., WIERSTRA, D., AND RIEDMILLER, M. 2014. Deterministic policy gradient algorithms. In *ICML*.

SOK, K. W., KIM, M., AND LEE, J. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph. 26*, 3, Article 107.

STADIE, B. C., LEVINE, S., AND ABBEEL, P. 2015. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

TAN, J., LIU, K., AND TURK, G. 2011. Stable proportional-derivative controllers. *Computer Graphics and Applications, IEEE 31*, 4, 34–44.

TAN, J., GU, Y., LIU, C. K., AND TURK, G. 2014. Learning bicycle stunts. *ACM Transactions on Graphics (TOG) 33*, 4, 50.

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

UCHIBE, E., AND DOYA, K. 2004. Competitive-cooperative-concurrent reinforcement learning with importance sampling. In *Proc. of International Conference on Simulation of Adaptive Behavior: From Animals and Animats*, 287–296.

VAN DER MAATEN, L., AND HINTON, G. E. 2008. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research 9*, 2579–2605.

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., GUEZ, A., AND SILVER, D. 2015. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*.

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.

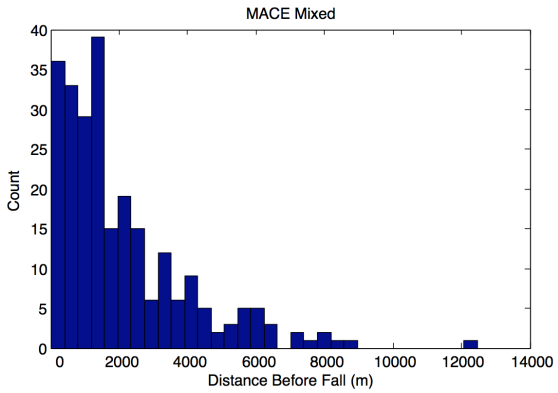
WIERING, M., AND VAN HASSELT, H. 2008. Ensemble algorithms in reinforcement learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 38*, 4, 930–936.

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

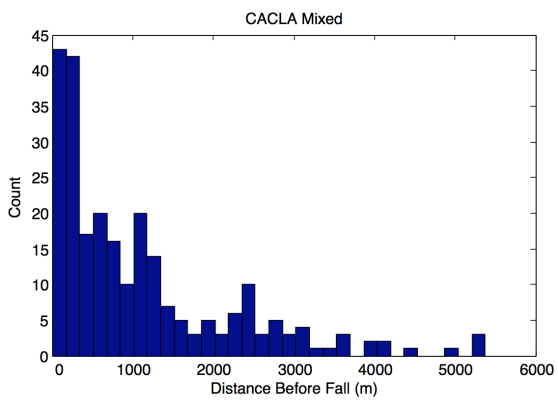
YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Transctions on Graphics 26*, 3, Article 105.

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.

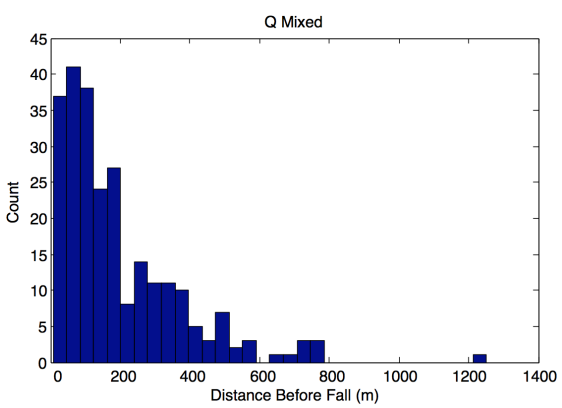
A Performance Histograms



(a) MACE mixed

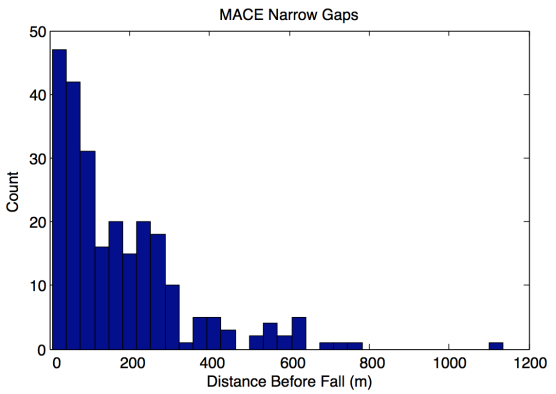


(b) CACLA mixed

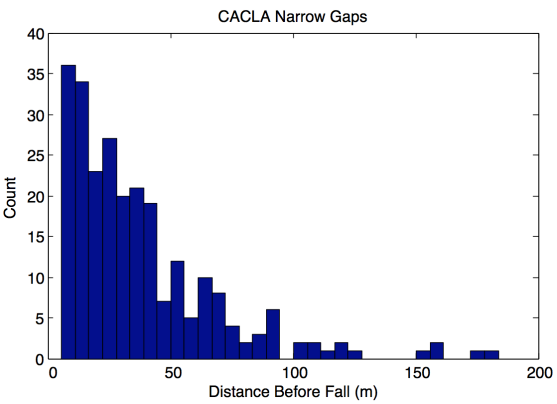


(c) Q mixed

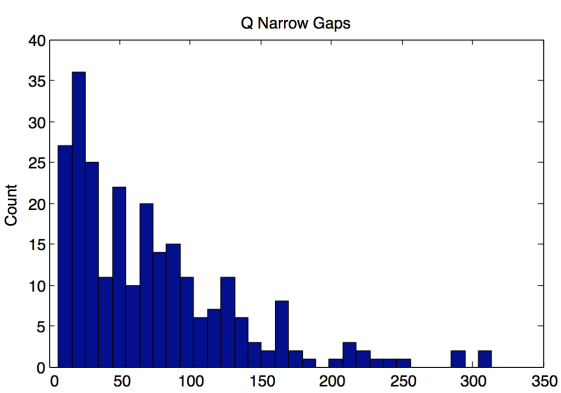
Figure 15: *Performance Histograms*



(a) MACE narrow-gaps

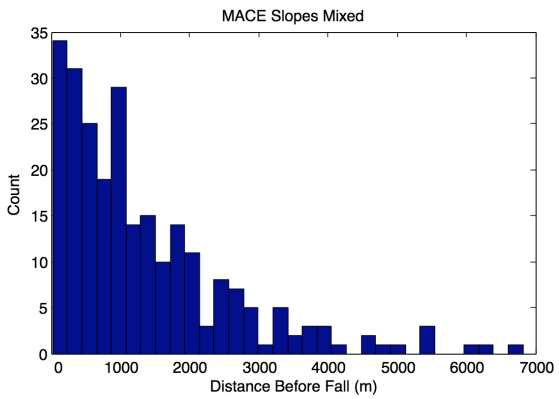


(b) CACLA narrow-gaps

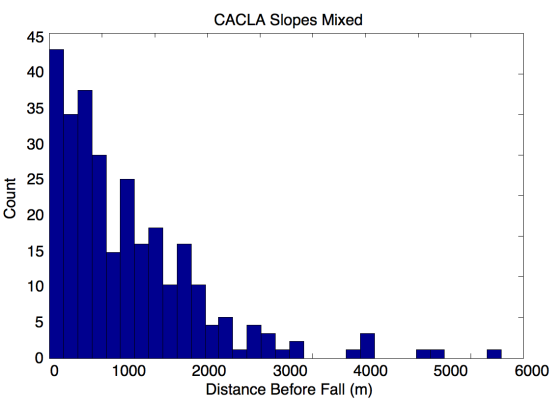


(c) Q narrow-gaps

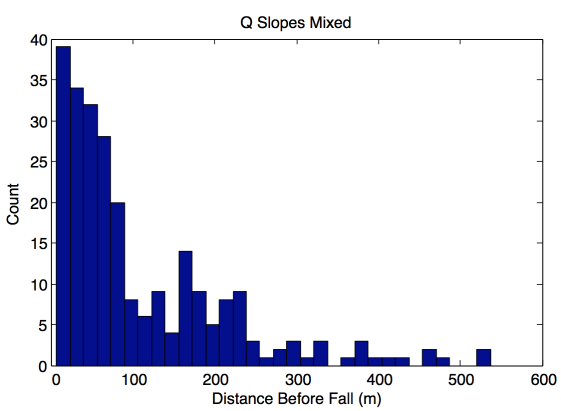
Figure 16: *Performance Histograms*



(a) MACE slopes-mixed

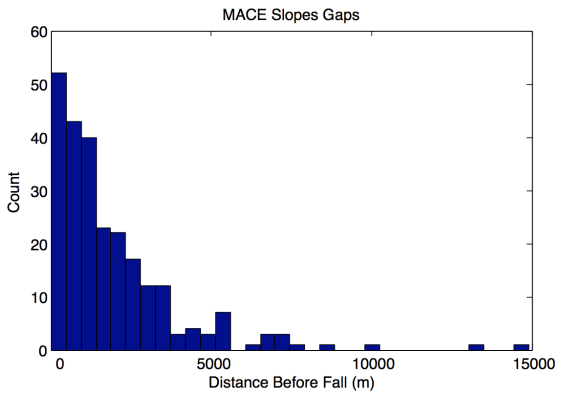


(b) CACLA slopes-mixed

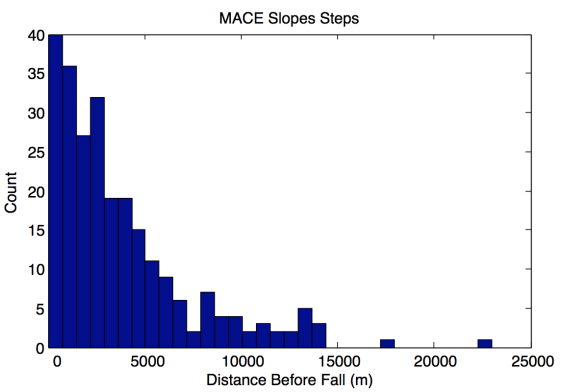


(c) Q slopes-mixed

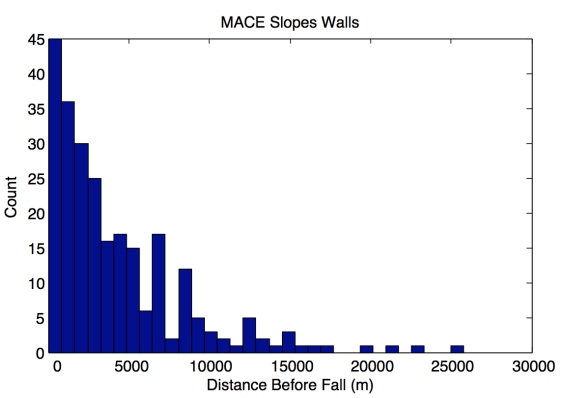
Figure 17: *Performance Histograms*



(a) MACE slopes-gaps

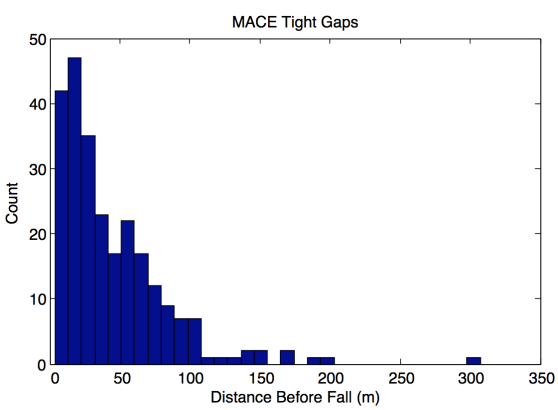


(b) MACE slopes-steps



(c) MACE slopes-walls

Figure 18: *Performance Histograms*



(a) MACE tight-gaps

Figure 19: *Performance Histograms*