DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning

XUE BIN PENG and GLEN BERSETH, University of British Columbia
KANGKANG YIN, National University of Singapore
MICHIEL VAN DE PANNE, University of British Columbia

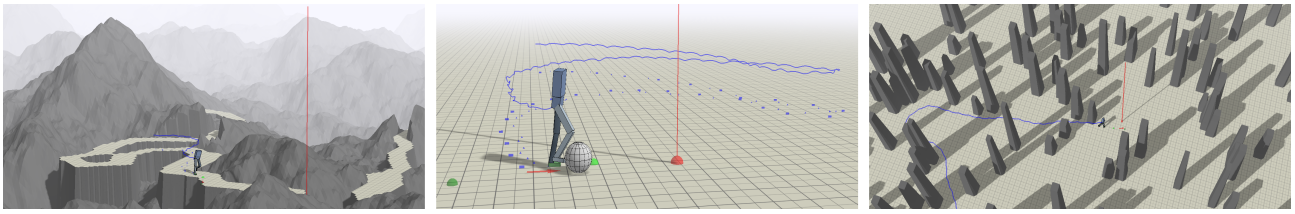

Fig. 1. Locomotion skills learned using hierarchical reinforcement learning. (a) Following a varying-width winding path. (b) Dribbling a soccer ball. (c) Navigating through obstacles.

Learning physics-based locomotion skills is a difficult problem, leading to solutions that typically exploit prior knowledge of various forms. In this paper we aim to learn a variety of environment-aware locomotion skills with a limited amount of prior knowledge. We adopt a two-level hierarchical control framework. First, low-level controllers are learned that operate at a fine timescale and which achieve robust walking gaits that satisfy stepping-target and style objectives. Second, high-level controllers are then learned which plan at the timescale of steps by invoking desired step targets for the low-level controller. The high-level controller makes decisions directly based on high-dimensional inputs, including terrain maps or other suitable representations of the surroundings. Both levels of the control policy are trained using deep reinforcement learning. Results are demonstrated on a simulated 3D biped. Low-level controllers are learned for a variety of motion styles and demonstrate robustness with respect to force-based disturbances, terrain variations, and style interpolation. High-level controllers are demonstrated that are capable of following trails through terrains, dribbling a soccer ball towards a target location, and navigating through static or dynamic obstacles.

CCS Concepts: • **Computing methodologies** → **Animation**; *Physical simulation*; *Control methods*; *Reinforcement learning*;

Additional Key Words and Phrases: physics-based character animation, motion control, locomotion skills

ACM Reference format:
Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.* 36, 4, Article 41 (July 2017), 16 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073602

DOI: http://dx.doi.org/10.1145/3072959.3073602

1 INTRODUCTION

Physics-based simulations of human skills and human movement have long been a promising avenue for character animation, but it has been difficult to develop the needed control strategies. While the learning of robust balanced locomotion is a challenge by itself, further complexities are added when the locomotion needs to be used in support of tasks such as dribbling a soccer ball or navigating among moving obstacles. Hierarchical control is a natural approach towards solving such problems. A low-level controller (LLC) is desired at a fine timescale, where the goal is predominately about balance and limb control. At a larger timescale, a high-level controller (HLC) is more suitable for guiding the movement to achieve longer-term goals, such as anticipating the best path through obstacles. In this paper, we leverage the capabilities of deep reinforcement learning (RL) to learn control policies at both timescales. The use of deep RL allows skills to be defined via objective functions, while enabling for control policies based on high-dimensional inputs, such as local terrain maps or other abundant sensory information. The use of a hierarchy enables a given low-level controller to be reused in support of multiple high-level tasks. It also enables high-level controllers to be reused with different low-level controllers.

Our principal contribution is to demonstrate that environment-aware 3D bipedal locomotion skills can be learned with a limited amount of prior structure being imposed on the control policy. In support of this, we introduce the use of a two-level hierarchy for deep reinforcement learning of locomotion skills, with both levels of the hierarchy using an identical style of actor-critic algorithm. To the best of our knowledge, we demonstrate some of the most capable dynamic 3D walking skills for model-free learning-based methods, i.e., methods that have no direct knowledge of the equations of motion, character kinematics, or even basic abstract features such as the center of mass, and no *a priori* control-specific feedback structure. Our method comes with its own limitations, which we also discuss.

2 RELATED WORK

Modeling movement skills, locomotion in particular, has a long history in computer animation, robotics, and biomechanics. It has also recently seen significant interest from the machine learning community as an interesting and challenging domain for reinforcement learning. Here we focus only on the most closely related physics-based work in computer animation and reinforcement learning.

Physics-based Character Control: A recent survey on physics-based character animation and control techniques provides a comprehensive overview of work in this area [Geijtenbeek and Pronost 2012]. An early and enduring approach to controller design has been to structure control policies around finite state machines (FSMs) and feedback rules that use a simplified abstract model or feedback law. These general ideas have been applied to human athletics and running [Hodgins et al. 1995] and a rich variety of walking styles [Coros et al. 2010; Lee et al. 2010; Yin et al. 2007]. Many controllers developed for physics-based animation further use optimization methods to improve controllers developed around an FSM-structure, or use an FSM to define phase-dependent objectives for an inverse dynamics optimization to be solved at each time step. Policy search methods, e.g., stochastic local search or CMA [Hansen 2006], can be used to optimize the parameters of the given control structures to achieve a richer variety of motions, e.g., [Coros et al. 2011; Yin et al. 2008], and efficient muscle-driven locomotion [Wang et al. 2009]. Policy search has been successfully applied directly to time-indexed splines and neural networks in order to learn a variety of bicycle stunts [Tan et al. 2014]. An alternative class of approach is given by trajectory optimization methods, which can compute solutions offline, e.g., [Al Borno et al. 2013], that can be adapted for online model-predictive control [Hämäläinen et al. 2015; Tassa et al. 2012], or that can compute optimized actions for the current time-step using quadratic programming, e.g., [de Lasa et al. 2010; Macchietto et al. 2009]. Wu and Popović [2010] proposed a hierarchical framework that incorporates a footstep planner and model-predictive control for bipedal locomotion across irregular terrain. To further improve motion quality and enrich the motion repertoire, data-driven models incorporate motion capture examples in constructing controllers, most often using a learned or model-based trajectory tracking method [da Silva et al. 2008; Liu et al. 2016, 2012; Muico et al. 2009; Sok et al. 2007].

Reinforcement Learning for Simulated Locomotion: The neuroanimator work [Grzeszczuk et al. 1998] stands out as an early demonstration of a form of direct policy gradient method using a recurrent neural network, as applied to 3D swimming movements. Recent years have seen a resurgence of effort towards tackling reinforcement learning problems for simulated agents with continuous state and continuous action spaces. Many of the example problems consist of agents in 2D and 3D physics-based simulations that learn to swim, walk, and hop. Numerous methods have been applied to tasks that include planar biped locomotion or planar swimming: trajectory optimization [Levine and Abbeel 2014; Levine and Koltun 2014; Mordatch and Todorov 2014]; trust region policy optimization (TRPO) [Schulman et al. 2015]; and actor-critic approaches [Lillicrap et al. 2015; Mnih et al. 2016; Peng and van de Panne 2016].

Progress has recently been made on the harder problem of achieving 3D biped locomotion using learned model-free policies without the use of *a priori* control structures. Tackling this problem *de novo*, i.e., without any hints as to what walking looks like, is particularly challenging. Recent work using generalized advantage estimation together with TRPO [Schulman et al. 2016] demonstrates sustained 3D locomotion for a biped with ball feet. Another promising approach uses trajectory optimization methods to provide supervision for a recurrent neural network to generate stepping movements for 3D bipeds and quadrupeds [Mordatch et al. 2015], and with the final motion coming from an optimization step rather than directly from a forward dynamics simulation. Hierarchical control of 3D locomotion control has been recently proposed [Heess et al. 2016], and is perhaps the closest work to our own. Low-level, high-frequency controllers are first learned during a pretraining phase, in conjunction with a provisional high-level controller with access to task-relevant information and that can communicate with the low-level controller via a modulatory signal. Once pretraining has been completed, the low-level controller structure is fixed and other high-level controllers can then be trained. The 3D humanoid is trained to navigate a slalom course, an impressive feat given the *de novo* nature of the locomotion learning. In our work we demonstrate:(a) significantly more natural locomotion and the ability to walk with multiple styles that can be interpolated; (b) locomotion that has significant (quantified) robustness; (c) the learning of controllers for four high-level tasks that use high-dimensional input, i.e., the ability to see the surroundings using ego-centric terrain maps; and (d) learning of a soccer-dribbling biped controller. In support of this, we use: a bilinear phase transform for the low-level controllers; the use of one or several reference motions; optional style reward terms; and the use of a two-step foot plan that the high-level controller can communicate to the low-level controller. We note that reference motions have been previously utilized to guide the training of neural network policies for 3D bipedal locomotion [Levine and Koltun 2013].

Learning of high-level controllers for physics-based characters has been successfully demonstrated for several locomotion and obstacle-avoidance tasks [Coros et al. 2009; Peng et al. 2015, 2016]. Alternatively, planning using a learned high-level dynamics model has also been proposed for locomotion tasks [Coros et al. 2008]. However, the low-level controllers for these learned policies are still designed with significant human insight, and the recent works of Peng et al. are demonstrated only for planar motions.

Motion Planning: Motion planning is a well-studied problem, which typically investigates how characters or robots should move in constrained environments. For wheeled robots, such problem can usually be reduced to finding a path for a point robot [Kavraki et al. 1996]. Motion planning for legged robots is significantly more challenging due to the increased degrees of freedom and tight coupling with the underlying locomotion dynamics. When quadrupeds are equipped with robust mobility control, a classic A^* path planner can be used to compute steering and forward speed commands to the locomotion controller to navigate in real-world environment with high success [Wooden et al. 2010]. However, skilled balanced motions are more difficult to achieve for bipeds and thus they are

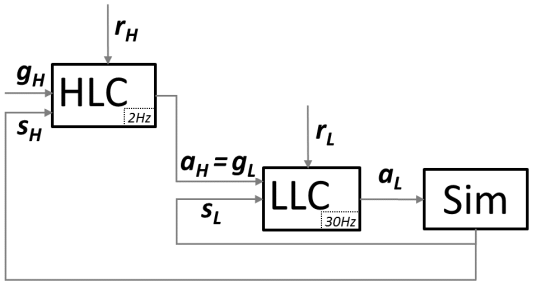

Fig. 2. System Overview

harder to plan and control [Kuffner et al. 2005]. Much of the work in robotics emphasizes footstep planning, e.g., [Chestnutt et al. 2005], with some work on full-body motion generation, e.g., [Grey et al. 2016]. Possibility graphs are proposed [Grey et al. 2016] to use high-level approximations of constraint manifolds to rapidly explore the possibility of actions, thereby allowing lower-level motion planners to be utilized more efficiently. Our hierarchical planning framework and the step targets produced by the HLC are partly inspired by this previous work from humanoid robotics.

Motion planning in support of character animation has been studied for manipulation tasks [Bai et al. 2012; Yamane et al. 2004] as well as full-body behaviours. The full-body behaviour planners often work with kinematic motion examples [Lau and Kuffner 2005; Lee and Lee 2004; PettrÃĭ et al. 2003]. Planning for physics-based characters is often achieved with the help of abstract dynamic models in low-dimensional spaces [Mordatch et al. 2010; Ye and Liu 2010]. A hybrid approach is adopted in [Liu et al. 2012] where a high-level kinematic planner directs the low-level dynamic control of specific motion skills.

3 OVERVIEW

An overview of the DeepLoco system is shown in Figure 2. The system is partitioned into two components that operate at different timescales. The high-level controller (HLC) operates at a coarse timescale of 2 Hz, the timescale of walking steps, while the low-level controller (LLC) operates at 30 Hz, the timescale of low-level control actions such as PD target angles. Finally, the physics simulation is performed at 3 kHz. Together, the HLC and LLC form a two-level control hierarchy where the HLC processes the high-level task goals g_H and provides the LLC with low-level intermediate goals g_L that direct the character towards fulfilling the overall task objectives. When provided with an intermediate goal from the HLC, the LLC coordinates the motion of the character's various joints in order to fulfill the intermediate goals. This hierarchical partitioning of control allows the controllers to explore behaviours spanning different spatial and temporal abstractions, thereby enabling more efficient exploration of task-relevant strategies.

The inputs to the HLC consist of the state, s_H, and the high-level goal, g_H, as specified by the task. It outputs an action, a_H, which then serves as the current goal g_L for the LLC. s_H provides both proprioceptive information of the character's configuration as well as exteroceptive information about its environment. In our framework, the high level action, a_H, consists of a footstep plan for the LLC.

The LLC receives the state, s_L, and an intermediate goal, g_L, as specified by the HLC, and outputs an action a_L. Unlike the high-level state s_H, s_L consists mainly of proprioceptive information describing the state of the character. The low-level action a_L specifies target angles for PD controllers positioned at each joint, which in turn compute torques that drive the motion of the character.

The actions from the LLC are applied to the simulation, which in turn produces updated states s_H and s_L by extracting the relevant features for the HLC and LLC respectively. The environment then also provides separate reward signals r_H and r_L to the HLC and LLC, reflecting progress towards their respective goals g_H and g_L. Both controllers are trained with a common actor-critic learning algorithm. The policy (actor) is trained using a positive-temporal difference update scheme modeled after CACLA [Van Hasselt 2012], and the value function (critic) is trained using Bellman backups.

4 POLICY REPRESENTATION AND LEARNING

Let $\pi(s, g) : S \times G \rightarrow A$ represent a deterministic policy, which maps a state $s \in S$ and goal $g \in G$ to an action $a \in A$, while a stochastic policy $\pi(s, g, a) : S \times G \times A \rightarrow \mathbb{R}$ represents the conditional probability distribution of a given s and g, $\pi(s, g, a) = p(a|s, g)$. For a particular s and g, the action distribution is modeled by a Gaussian $\pi(s, g, a) = G(\mu(s, g), \Sigma)$, with a parameterized mean $\mu(s, g)$ and fixed covariance matrix Σ. Each policy query in turn samples an action from the distribution according to

$$a = \mu(s, g) + \mathcal{N}, \quad \mathcal{N} \sim G(0, \Sigma) \tag{1}$$

generated by applying Gaussian noise to the mean action $\mu(s, g)$. While the covariance $\Sigma = diag(\{\sigma_i\})$ is represented by manually-specified values $\{\sigma_i\}$ for each action parameter, the mean is represented by a neural network $\mu(s, g|\theta)$ with parameters θ. Large values of $\{\sigma_i\}$ can cause excessively noisy motions, which are prone to falling, while small values can lead to slow learning. We found that setting $\sqrt{\sigma_i}$ to approximately 10% of the allowed range of values for each joint to be effective in practice.

During training, a stochastic policy enables the character to explore new actions that may prove promising, but the addition of exploration noise can impact performance at runtime. Therefore, at runtime, a deterministic policy, which always selects the mean action $\pi(s, g) = \mu(s, g)$, is used instead. The choice between a stochastic and deterministic policy can be denoted by the addition of a binary indicator variable $\lambda \in \{0, 1\}$

$$a = \mu(s, g) + \lambda\mathcal{N} \tag{2}$$

where 1 indicates a stochastic policy with added exploration noise, and 0 a deterministic policy that always selects the mean action. During training, ϵ-greedy exploration can be incorporated by randomly enabling and disabling exploration noise according to a Bernoulli distribution $\lambda \sim \text{Ber}(\epsilon)$, where ϵ represents the probability of action exploration by applying noise to the mean action.

In reinforcement learning, the objective is often to learn an optimal policy π^* that maximizes the expected long term cumulative reward $J(\pi)$, expressed as the discounted sum of immediate rewards $r_t \in \mathbb{R}$ with discount factor $\gamma \in [0, 1]$.

$$J(\pi) = \mathbb{E}_{r_0, r_1, \ldots r_T}[r_0 + \gamma r_1 + \ldots + \gamma^T r_T|\pi] \tag{3}$$

where T is a horizon that may be infinite. The reward function $r_t = r(s_t, g_t, a_t)$ provides the agent with feedback regarding the desirability of performing action a_t at state s_t given goal g_t. The reward function is therefore an interface through which users can shape the behaviour of the agent by assigning higher rewards to desirable behaviours, and lower rewards to less desirable ones. If π is modeled as a parametric function with parameters θ, then the expected cumulative reward can be re-expressed as $J(\theta)$, and the goal of learning π^* can be formulated as finding the optimal set of parameters θ^*

$$\theta^* = \arg\max_\theta J(\theta) \tag{4}$$

Policy gradient methods are a popular family of algorithms for solving this class of problems [Sutton et al. 2000]. These methods perform gradient ascent on the objective using empirical estimates of the policy gradient $\nabla_\theta J(\theta)$, i.e. the gradient of $J(\theta)$ with respect to the policy parameters θ. This class of methods lies at the heart of our framework.

Algorithm 1 illustrates the common learning algorithm for both the LLC and HLC. For the purpose of learning, the character's experiences are summarized by tuples $\tau_i = (s_i, g_i, a_i, r_i, s_i', \lambda_i)$, recording the start state, goal, action, reward, next state, and application of exploration noise for each action performed by the character. The tuples are stored in an experience replay memory D and used to update the policy. Each policy is trained using an actor-critic framework, where a policy $\pi(s, g, a|\theta_\mu)$ and value function $V(s, g|\theta_v)$, with parameters θ_μ and θ_v, are learned in tandem. The value function is trained to predict the expected cumulative reward of following the policy starting at a given state s and goal g. To update the value function, a minibatch of n tuples $\{\tau_i\}$ are sampled from D and used to perform a Bellman backup

$$y_i \leftarrow r_i + \gamma V(s_i', g_i|\theta_v) \tag{5}$$

$$\theta_v \leftarrow \theta_v + \alpha_v \left(\frac{1}{n} \sum_i \nabla_{\theta_v} V(s_i, g_i|\theta_v)(y_i - V(s_i, g_i|\theta_v)) \right) \tag{6}$$

with α_v being the critic stepsize. The learned value function is then used to update the policy. Policy improvement is performed using a CACLA-style positive temporal difference update [Van Hasselt 2012]. Since the policy gradient as defined above is for stochastic policies, policy updates are performed using only tuples with added exploration noise (i.e. $\lambda_i = 1$).

$$\delta_i \leftarrow r_i + \gamma V(s_i', g_i|\theta_v) - V(s_i, g_i|\theta_v) \tag{7}$$

if $\delta_i > 0$:

$$\theta_\mu \leftarrow \theta_\mu + \alpha_\mu \left(\frac{1}{n} \nabla_{\theta_\mu} \mu(s_i, g_i|\theta_\mu) \Sigma^{-1} (a_i - \mu(s_i, g_i|\theta_\mu)) \right) \tag{8}$$

where α_μ is the actor stepsize. Equation 8 can be interpreted as a stochastic gradient ascent step along an estimate of the policy gradient for a Gaussian policy.

ALGORITHM 1: Actor-Critic Algorithm Using Positive Temporal Difference Updates

$\theta_\mu \leftarrow$ random weights
$\theta_v \leftarrow$ random weights
while not done **do**
  for $step = 1, \ldots, m$ **do**
    $s \leftarrow$ start state
    $g \leftarrow$ goal
    $\lambda \leftarrow \text{Ber}(\epsilon_t)$
    $a \leftarrow \mu(s, g|\theta_\mu) + \lambda \mathcal{N}, \quad \mathcal{N} \sim G(0, \Sigma)$
    Apply a and simulate forward one step
    $s' \leftarrow$ end state
    $r \leftarrow$ reward
    $\tau \leftarrow (s, g, a, r, s', \lambda)$
    store τ in D
  end for

  Update value function:
  Sample minibatch of n tuples $\{\tau_i = (s_i, g_i, a_i, r_i, s_i', \lambda_i)\}$ from D
  for each τ_i **do**
    $y_i \leftarrow r_i + \gamma V(s_i', g_i|\theta_v) - V(s_i, g_i|\theta_v)$
  end for
  $\theta_v \leftarrow \theta_v + \alpha_v \left(\frac{1}{n} \sum_i \nabla_{\theta_v} V(s_i, g_i|\theta_v)(y_i - V(s_i, g_i|\theta_v)) \right)$

  Update policy:
  Sample minibatch of n tuples $\{\tau_j = (s_j, g_j, a_j, r_j, s_j', \lambda_j)\}$ from D where $\lambda_j = 1$
  for each τ_j **do**
    $\delta_j \leftarrow r_j + \gamma V(s_j', g_j|\theta_v) - V(s_j, g_j|\theta_v)$
    if $\delta_j > 0$ **then**
      $\triangle a_j \leftarrow a_j - \mu(s_j, g_j|\theta_\mu)$
      $\theta_\mu \leftarrow \theta_\mu + \alpha_\mu \left(\frac{1}{n} \nabla_{\theta_\mu} \mu(s_j, g_j|\theta_\mu) \Sigma^{-1} \triangle a_j \right)$
    end if
  end for
end while

5 LOW-LEVEL CONTROLLER

The low-level controller LLC is responsible for coordinating joint torques to mimic the overall style of a reference motion while satisfying footstep goals and maintaining balance. The reference motion is represented by keyframes that specify target poses at each timestep t. The LLC is queried at 30Hz, where each query provides as input the

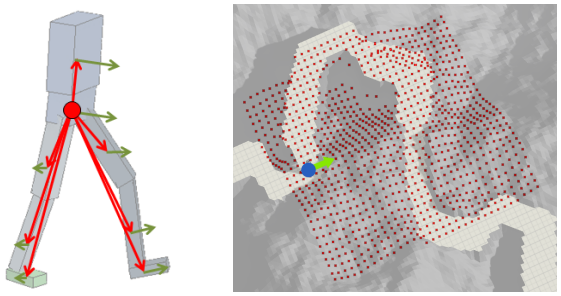

Fig. 3. **left:** The character state features consist of the positions of each link relative to the root **(red arrows)**, their rotations, linear velocities **(green arrows)**, and angular velocities. **right:** The terrain features consist of a 2D heightmap of the terrain sampled on a regular grid. All heights are expressed relative to height of the ground immediately under the root of the character. The heightmap has a resolution of 32x32 and occupies an area of approximately 11x11m.

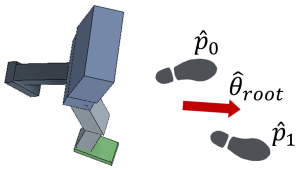

Fig. 4. The goal g_L for the LLC is represented as a footstep plan, specifying the target positions $\hat{p}_0$ and $\hat{p}_1$ for the next two steps, and the target heading for the root $\hat{\theta}_{root}$.

state s_L, representing the character state, and goal g_L, representing a footstep plan. The LLC then produces an action a_L specifying PD target angles for every joint, relative to their parent link.

LLC State: The LLC input state s_L, shown in Figure 3 (left), consists mainly of features describing the character's configuration. These features include the center of mass positions of each link relative to the character's root, designated as the pelvis, their relative rotations with respect to the root expressed as quaternions, and their linear and angular velocities. Two binary indicator features are included, corresponding to the character's feet. The features are assigned 1 when their respective foot is in contact with the ground and 0 otherwise. A phase variable $\phi \in [0, 1]$ is also included as an input, which indicates the phase along the gait cycle. Each gait cycle has a fixed period of 1 s, corresponding to 0.5 s per step. The phase variable advances at a fixed rate and helps keep the LLC in sync with the reference motion. For the reference motions, the phase is linearly interpolated between foot contact times. Combined, the state features create a 110D state space.

LLC Goal: Each footstep plan $g_L = (\hat{p}_0, \hat{p}_1, \hat{\theta}_{root})$, as shown in Figure 4, specifies the 2D target position $\hat{p}_0$ relative to the character on the horizontal plane for the swing foot at the end of the next step, as well as the target location for the following step $\hat{p}_1$. This is motivated by work showing that "two steps is enough" [Zaytsev et al. 2015]. In addition to target step positions, the footstep plan also provides a desired heading $\hat{\theta}_{root}$ for the root of the character for the immediate next step.

LLC Action: The action a_L produced by the LLC specifies target positions for PD controllers positioned at each joint. The target joint positions are represented in 4 dimensional axis-angle form, with axis normalization occurring when applying the actions, i.e., the output action from the network need not be normalized. Each action parameter is clamped to stay within permissible values depending on the range of motion of each joint. This yields a 22D action space.

5.1 Reference Motion

A reference motion (or set of motions) serves to help specify the desired walking style, while also helping to guide the learning. The reference motion can be a single manually keyframed motion cycle, or one or more motion capture clips. The goal for the LLC is to mimic the overall style of the reference motion rather than precisely tracking it. The reference motion will generally not satisfy the desired footstep goals, and is often not physically realizable in any case because of the approximate nature of a hand-animated gait cycle, or model mismatches in the case of a motion capture clip. At each timestep t a reference motion provides a reference pose $\hat{q}(t)$ and reference velocity $\hat{\dot{q}}(t)$, computed via finite-differences $\hat{\dot{q}}(t) \approx \frac{\hat{q}(t+\triangle t)-\hat{q}(t)}{\triangle t}$. The use of multiple reference motion clips can help produce better turning behaviors, as best seen in the supplemental video. To make use of multiple reference motions, we construct a kinematic controller $\hat{q}^*(\cdot) \leftarrow K(s, g_L)$, when given the simulated character state s and a footstep plan g_L, selects the appropriate motion from a small set of motion clips that best realizes the footstep plan g_L. To construct the set of reference motions for the kinematic controller, we segmented 7 s of motion capture data of walking and turning motions, into individual clips $\hat{q}^j(\cdot)$, each corresponding to a single step. A step begins on the stance foot heel-strike and ends on the swing foot heel-strike. Each clip is preprocessed to be in right stance, and linear time-warping is applied to normalize the step duration to 0.5 s. During training, the reference motions are mirrored as necessary to be consistent with the simulated character's stance leg. To help synchronize the reference motion with the simulated character, we define the phase ϕ of a motion as a linear function with 0 at the start of a step and 1 at the end. A vector of features $\psi(\hat{q}^j(\cdot)) = (p_{stance}, p_{swing}, \theta_{root})$ are then extracted for each clip and later used to select the appropriate clip for a given query. The features include the stance foot position p_{stance} at the start of a clip, the swing foot position p_{swing} at the end of the clip, and the root orientation θ_{root} on the horizontal plane at the end of the clip. All features are expressed with respect to the character's local coordinate frame at the start of each clip, with the origin beneath the character's root and the x-axis aligned along the root's heading direction. The mocap clips were collected from http://animation.comp.nus.edu.sg/nusmocap.html. Besides segmenting and retargetting the clips for our character, no additional processing was performed.

During training, $K(s, g_L)$ is queried at the beginning of each step to select the reference clip for the upcoming step. To select among the motion clips, a similar set of features $\psi(s, g_L)$ are extracted from s and g_L, where p_{stance} is specified by the stance foot position from the simulated character state s, p_{swing} and θ_{root} are specified by the target footstep position $\hat{p}_0$ and root orientation $\hat{\theta}_{root}$ from g_L. The most suitable clip is then selected according to:

$$K(s, g_L) = \arg\min_{\hat{q}^j(\cdot)} ||\psi(s, g_L) - \psi(\hat{q}^j(\cdot))|| \tag{9}$$

The selected clip then acts as the reference motion to shape the reward function for the LLC over the course of the upcoming step.

5.2 LLC Reward

Given the reference pose $\hat{q}(t)$ and velocity $\hat{\dot{q}}(t)$ the LLC reward r_L is defined as a weighted sum of objectives that encourage the character to imitate the style of the reference motion while following the footstep plan,

$$\begin{aligned} r_L = w_{pose} r_{pose} + w_{vel} r_{vel} + w_{root} r_{root} \\ + w_{com} r_{com} + w_{end} r_{end} + w_{heading} r_{heading} \end{aligned} \tag{10}$$

using $(w_{pose}, w_{vel}, w_{root}, w_{com}, w_{end}, w_{heading}) = (0.5, 0.05, 0.1, 0.1, 0.2, 0.1)$. r_{pose}, r_{vel}, r_{root}, and r_{com} encourages the policy to reproduce the given reference motion, while r_{end} and

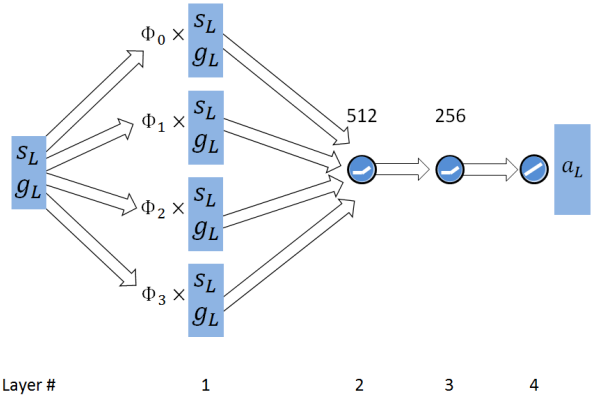

Fig. 5. Schematic illustration of the LLC network. The input consists of the state s_L and goal g_L. The first layer applies the bilinear phase transform and the resulting features are processed by a series of fully-connected layers. The output layer produces the action a_L, which specifies PD targets for each joint.

$r_{heading}$ encourages it to follow the footstep plan.

$$r_{pose} = \exp\left(-\sum_i w_i d(\hat{q}_i(t), q_i)^2\right) \tag{11}$$

where q_i represents the quaternion rotation of joint i and $d(\cdot,\cdot)$ computes the distance between two quaternions. w_i are manually specified weights for each joint. Details for the other reward terms are available in the supplemental material. We choose to keep rewards constrained to $r \in [0, 1]$.

5.3 Bilinear Phase Transform

While the phase variable ϕ helps to keep the LLC in sync with the reference motion, in our experiments this did not appear to be sufficient for the network to clearly distinguish the different phases of a walk, often resulting in foot-dragging artifacts. To help the network better distinguish between different phases of a motion, we take inspiration from bilinear pooling models for vision tasks [Fukui et al. 2016]. From the scalar phase variable ϕ we construct a tile-coding $\Phi = (\Phi_0, \Phi_1, \Phi_2, \Phi_3)^T$, where $\Phi_i \in \{0, 1\}$ is 1 if ϕ lies within its phase interval and 0 otherwise. For example, $\Phi_0 = 1$ iff $0 \leq \phi < 0.25$, and $\Phi_1 = 1$ iff $0.25 \leq \phi < 0.5$, etc. Given the original input vector (s_L, g_L), the bilinear phase transform computes the outer product

$$\begin{pmatrix} s_L \\ g_L \end{pmatrix} \Phi^T = \left[\Phi_0 \begin{pmatrix} s_L \\ g_L \end{pmatrix}, \Phi_1 \begin{pmatrix} s_L \\ g_L \end{pmatrix}, \Phi_2 \begin{pmatrix} s_L \\ g_L \end{pmatrix}, \Phi_2 \begin{pmatrix} s_L \\ g_L \end{pmatrix}\right] \tag{12}$$

which is then processed by successive layers of the network. This representation results in a feature set where only a sparse subset of the features, corresponding to the current phase interval, are active at a given time. This effectively encodes a prior into the network that different behaviours are expected at different phases of the motion. Note that the scalar phase variable ϕ is still included in s_L to allow the LLC to track its progress within each phase interval.

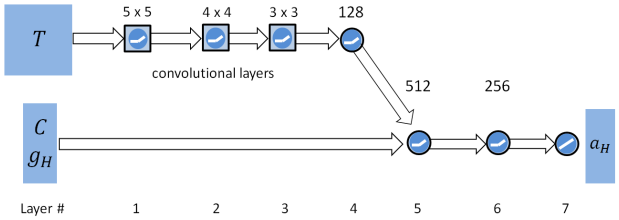

Fig. 6. Schematic illustration of the HLC network. The input consists of a terrain map T, character features C, and goal g_H. The output action a_H specifies a footstep plan g_L for the LLC.

5.4 LLC Network

A schematic diagram of the LLC network is shown in Figure 5. The LLC is represented by a 4-layered neural network that receives as input s_L and g_L, and outputs the mean $\mu(s_L, g_L)$ of the action distribution. The first layer applies the bilinear phase transform to the inputs, and the resulting bilinear features are then processed by two fully-connected layers with 512 and 256 units each. ReLU activation functions are applied to both hidden layers [Nair and Hinton 2010]. Finally a linear output layer computes the mean action. The LLC value function $V_L(s_L, g_L)$ is modeled by a similar network, but with a single linear unit in the output layer. Each LLC network has approximately 500k parameters.

5.5 LLC Training

LLC training proceeds episodically where the character is initialized to a default pose at the beginning of each episode. An episode is simulated for a maximum of 200 s but is terminated early if the character falls, leaving the character with 0 reward for the remainder of the episode. A fall is detected when the torso of the character makes contact with the ground. At the beginning of each walking step, a new footstep plan $g_L^k = (\hat{p}_0^k, \hat{p}_1^k, \hat{\theta}_{root}^k)$ is generated by randomly adjusting the previous plan g_L^{k-1} according to

$$\begin{aligned} \hat{p}_0^k &= \hat{p}_1^{k-1} \\ \hat{\theta}_{root}^k &= \hat{\theta}_{root}^{k-1} + \mathcal{N}, \quad \mathcal{N} \sim G(0, 0.25^2) \\ \hat{p}_1^k &= \hat{p}_0^k + \triangle p(\hat{\theta}_{root}^k) \end{aligned} \tag{13}$$

where $\triangle p(\hat{\theta}_{root}^k)$ advances the step position along the heading direction $\hat{\theta}_{root}^k$ by a fixed step length of 0.4 m to obtain a new target step position.

After a footstep plan has been determined for the new step, the kinematic controller $K(s_L, g_L)$ is queried for a new reference motion. The reference motion $\hat{q}(\cdot)$ is then used by the reward function for the duration of the step, which guides the LLC towards a stepping motion that approximately achieves the desired footstep goal g_L.

5.6 Style Modification

In addition to imitating a reference motion, the LLC can also be stylized by simple modifications to the reward function. In the following examples, we consider the addition of a style term c_{style}

to the pose reward r_{pose}.

$$r_{pose} = \exp\left(-\sum_i w_i d(\hat{q}_i(t), q_i)^2 - w_{style} c_{style}\right) \quad (14)$$

where c_{style} provides an interface through which the user can shape the motion of the LLC. w_{style} is a user-specified weight that trades off between conforming to the reference motion and satisfying the desired style.

Forward/Sideways Lean: By using c_{style} to specify a desired waist orientation, the LLC can be steered towards learning a robust walk while leaning forward or sideways.

$$c_{style} = d(\hat{q}(t)_{waist}, q_{waist})^2 \quad (15)$$

where $\hat{q}(\cdot)_{waist}$ is a quaternion specifying the desired waist orientation.

Straight Leg(s): Similarly, c_{style} can be used to penalize bending of the knees, resulting in a locked-knee walk.

$$c_{style} = d(q_I, q_{knee})^2 \quad (16)$$

with q_I being the identity quaternion. Using this style term, we trained two LLC's, one with the right leg encouraged to be straight, and one with both legs straight.

High-Knees: A high-knees walk can be created by using c_{style} to encourage the character to lift its knees higher during each step,

$$c_{style} = (\hat{h}_{knee} - h_{knee})^2 \quad (17)$$

where $\hat{h}_{knee} = 0.8m$ is the target height for the swing knee with respect to the ground.

In-place Walk: By replacing the reference motion $\hat{q}(\cdot)$ with a single hand-authored clip of an in-place walk, the LLC can be trained to step in-place.

Separate networks are trained for each stylized LLC by bootstrapping from the nominal walk LLC. The weights of each network are initialized from those of the nominal walk, then fine-tuned using the stylized reward functions. Furthermore, we show that it is possible to interpolate different stylized LLC's while also remaining robust. Let $\pi_L^a(s_L, g_L)$ and $\pi_L^b(s_L, g_L)$ represent LLC's trained for style a and b. A new LLC $\pi_L^c(s, g)$ can be defined by linearly interpolating the outputs of the two LLC's

$$\pi_L^c(s_L, g_L) = (1-u)\pi_L^a(s_L, g_L) + u\pi_L^b(s_L, g_L) \quad (18)$$

with $u \in [0, 1]$, allowing the character to seamlessly transition between the different styles. As shown in the results, we can also allow for moderate extrapolation.

6 HIGH-LEVEL CONTROLLER

While the LLC is primarily responsible for low-level coordination of the character's limbs for locomotion, the HLC is responsible for high-level task-specific objectives such as navigation. The HLC is queried at 2 Hz, corresponding to the beginning of each step. Every query provides as input a state s_H and a task-specific goal g_H. The HLC output action a_H specifies a footstep plan g_L for the LLC. The role of the HLC is therefore to provide intermediate goals for the LLC in order to achieve the overall task objectives.

HLC State: Unlike s_L, which provides mainly proprioceptive information describing the configuration of the character, s_H includes both proprioceptive and exteroceptive information describing the character and its environment. Each state $s_H = (C, T)$, consists of a set of character features C and terrain features T, shown in Figure 3 (right). C shares many of the same features as the LLC state s_L, but excludes the phase and contact features. T is represented by a 32×32 heightmap of the terrain around the character. The heightmap is sampled on a regular grid with an area of approximately 11×11 m. The samples extend 10 m in front of the character and 1 m behind. Example terrain maps are shown in Figure 7. The combined features result in a 1129D state space.

6.1 HLC Training

As with the LLC training, the character is initialized to a default pose at the start of each episode. Each episode terminates after 200 s or when the character falls. At the start of each step, the HLC is queried to sample an action a_H from the policy, which is then applied to the LLC as a footstep goal g_L. The LLC is executed for 0.5 s, the duration of one step, and an experience tuple τ is recorded for the step. Note that the weights for the LLC are frozen and only the HLC is being trained. Therefore, once trained, the same LLC can be applied to a variety of tasks by training task-specific HLC's that specify the appropriate intermediate footstep goals.

6.2 HLC Network

A schematic diagram of the HLC network is available in Figure 6. The HLC is modeled by a deep convolutional neural network that receives as input the state $H = (C, T)$ and task-specific goal g_H, and the output action $a_H = g_L$ specifies a footstep plan for the LLC for a single step. The terrain map T is first processed by a series of three convolutional layers, with 16 5×5 filters, 32 4×4 filters, and 32 3×3 filters respectively. The features maps from the final convolutional layer are processed by 128 fully-connected units. The resulting feature vector is concatenated with C and g_H, and processed by two additional fully-connected layers with 512 and 256 units. ReLUs are used for all hidden layers. The linear output layer produces the final action. Each HLC network has approximately 2.5 million parameters.

6.3 HLC Tasks

Path Following: In this task an HLC is trained to navigate narrow paths carved into rocky terrain. A random path of uniform height

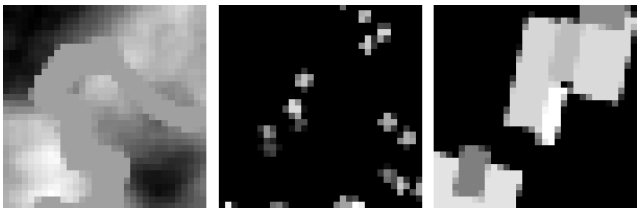

Fig. 7. 32×32 height maps are included as input features to the HLC. Each map covers a 11×11 m area. Values in the images are normalized by the minimum and maximum height within each map. **left:** path; **middle:** pillar obstacles; **right:** block obstacles.

is embedded into terrain generated using Perlin Noise [Perlin 2002]. The path width varies between 1 m and 2 m. A target location is placed randomly along the path, and advances along the path when the character is within 1 m of the target. The HLC goal $g_H = (\theta_{tar}, d_{tar})$ is represented by the direction to the target θ_{tar} relative to the character's facing direction, and the distance d_{tar} to the target on the horizontal plane. Since the policy is not provided with an explicit parameterization of the path as input, it must learn to recognize the path from the terrain map T and plan its footsteps accordingly.

The reward for this task is designed to encourage the character to move towards the target at a desired speed.

$$r_H = \exp\left(-\left(\min(0, u_{tar}^T v_{com} - \hat{v}_{com})\right)^2\right) \tag{19}$$

where v_{com} is the agent's centre of mass velocity on the horizontal plane, and u_{tar} is a unit vector on the horizontal plane pointing towards the target. $\hat{v}_{com} = 1\ m/s$ specifies the desired speed at which the character should move towards the target. The reward was designed to only penalize slower than desired motions, but can be easily modified to also penalize faster than desired motions.

Soccer Dribbling: Dribbling is a challenging task requiring both high-level and low-level planning. The objective is to move a ball to a target location, where the initial ball and target locations are randomly set at the beginning of each episode. The ball has a radius of 0.2 m and a mass of 0.1 kg. Having to learn a proper sequence of sub-tasks in the correct order makes this task particularly challenging. The agent must first move to the ball, and once it has possession of the ball, dribble the ball towards the target. When the ball has arrived at the target, the agent must then learn to stop moving the ball to avoid kicking the ball past the target. Since the policy does not have direct control over the ball, it must rely on complex contact dynamics in order to manipulate the ball. Furthermore, considering the LLC was not trained with motion data comparable to dribbling, the HLC has to learn to provide the appropriate footstep plans in order to elicit the necessary LLC behaviour. The goal $g_H = (\theta_{tar}, d_{tar}, \theta_{ball}, d_{ball}, h_{ball}, v_{ball}, \omega_{ball})$ consists of the target direction relative to the ball θ_{tar}, distance between the target and ball d_{tar}, ball direction relative to the agent's root θ_{ball}, distance between the ball and the agent d_{ball}, height of the ball's center of mass from the ground h_{ball}, the ball's linear velocity v_{ball}, and angular velocity ω_{ball}. Because the dribbling task occurs on a flat plane, the terrain map T is excluded from the HLC inputs, and the convolutional layers are removed from the network.

The reward for the soccer task consists of a weighted sum of terms which encourages the agent to move towards the ball r_{cv}, stay close to the ball r_{cp}, move the ball towards the target r_{bv}, and keep the ball close to the target r_{bp}.

$$r_H = w_{cv} r_{cv} + w_{cp} r_{cp} + w_{bv} r_{bv} + w_{bp} r_{bp} \tag{20}$$

with weights $(w_{cv}, w_{cp}, w_{bv}, w_{bp}) = (0.17, 0.17, 0.33, 0.33)$. Details for each term are available in the supplementary material.

Pillar Obstacles: A more common task is to traverse a reasonably dense area of static obstacles. Similar to the path following task, the objective is to reach a randomly placed target location. However, unlike the path following task, there exists many possible paths to reach the target. The HLC is therefore responsible for planning and steering the agent along a particular path. When the agent reaches the target, the target location is randomly changed. The base of each obstacle measures $0.75 \times 0.75\ m$, with height varying between 2 m and 8 m. Each environment instance is generated by randomly placing obstacles at the start of each episode. The goal g_H and reward function are the same as those used for the path following task.

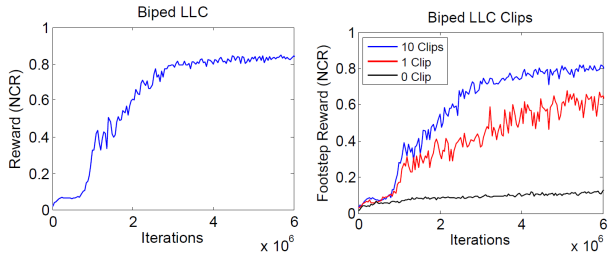

Fig. 8. **left:** Learning curve for the LLC. The network is randomly initialized and trained to mimic a nominal walk while following randomly generated footstep plans. **right:** learning curves for policies trained with 10 mocap clips, 1 hand-authored clip, and no motion clips.

Block Obstacles: This environment is a variant of the pillar obstacles environment, where the obstacles consist of large blocks with side lengths varying between 0.5 m and 7 m. The policy therefore must learn to navigate around large obstacles to find paths leading to the target location.

Dynamic Obstacles: In this task, the objective is to navigate across a dynamically changing environment in order to reach a target location. The environment is populated with obstacles moving at fixed velocities back and forth along randomly oriented linear paths. The velocities vary from 0.2 m/s to 1.3 m/s, with the agent's maximum velocity being approximately 1 m/s. Given the presence of dynamic obstacles, rather than using a heightfield as input, the policy is provided with a velocity-map. The environment is sampled for moving obstacles where each sample records the 2D velocity along the horizontal plane if a sample overlaps with an obstacle. If a sample point does not contain an obstacle, then the velocity is recorded as 0. The goal features and reward function are identical to those used in the path following task. This example should not be confused with a multiagent simulation because the moving obstacles themselves are not reactive.

7 RESULTS

The motions from the policies are best seen in the supplemental videos. We learn locomotion skills for a 3D biped, as modeled by eight links: three links for each leg and two links for the torso. The biped is 1.6 m tall and has a mass of 42 kg. The knee joints have one degree of freedom (DOF) and all other joints are spherical, i.e., three DOFs. We use a ground friction of $\mu = 0.9$. The character's motion is driven by internal joint torques from stable PD controllers [Tan et al. 2011] and simulated using the Bullet physics engine [Bullet 2015] at 3000 Hz. The k_p gains for the PD controllers are (1000,

LLC	Forward	Side	Incline	Decline
Nominal Walk	200N	210N	16%(9.1°)	11%(6.3°)
High-Knees	140N	190N	9%(5.1°)	5%(2.9°)
Straight Leg	150N	180N	12%(6.8°)	6%(3.4°)
Straight Legs	90N	130N	9%(5.1°)	5%(2.9°)
Forward Lean	180N	290N	10%(5.7°)	16%(9.1°)
Sideways Lean	160N	220N	7%(4.0°)	16%(9.1°)

Table 1. Maximum forwards and sideways push, and steepest incline and decline each LLC can tolerate before falling. Each push is applied for 0.25 s.

300, 300, 100) Nm/rad for the (waist, hip, knee, ankle), respectively. Derivative gains are specified as $k_d = 0.1k_p$. Torque limits are (200, 200, 150, 90) Nm, respectively. Joint limits are also in effect for all joints. All neural networks are built and trained with Caffe [Jia et al. 2014]. The values of the input states and output actions of the networks are normalized to range approximately between [-1, 1] using manually-specified offsets and scales. The output of the value network is normalized to be between [0, 1] by multiplying the cumulative reward by $(1 - \gamma)$. This normalization helps to ensure reasonable gradient magnitudes during backpropagation. Once trained, all results run faster than real-time.

LLC reference motions: We train controllers using a single planar keyframed motion cycle as a motion style to imitate, as well as a set of ten motion capture steps that correspond to approximately 7 s of data from a single human subject. The clips consist of walking motions with different turning rates. The character was designed to have similar measurements to those of the human subject. By default, we use the results based on the motion capture styles, as they allow for sharper turns and produce a moderate improvement in motion quality. Please see the supplementary video for a direct comparison.

Hyperparameter settings: Both LLC and HLC training share similar hyperparameter settings. Batches of $m = 32$ are collected before every update. The experience replay memory D records the 50k most recent tuples. Updates are performed by sampling minibatches of $n = 32$ tuples from D and applying stochastic gradient descent with momentum, with value function stepsize $\alpha_V = 0.01$, policy stepsize $\alpha_\mu = 0.001$, and momentum 0.9. L2 weight decay of 0.0005 is applied to the policy, but none is applied to the value function. Both the LLC and HLC use a discount factor of $\gamma = 0.95$. For the LLC, the ϵ-greedy exploration rate ϵ_t is initialized to 1 and linearly annealed to 0.2 over 1 million iterations. For the HLC, ϵ_t is initialized to 1 and annealed to 0.5 over 200k iterations. The LLC is trained for approximately 6 million iterations, requiring about 2 days of compute time on a 16-core cluster using a multithreaded C++ implementation. Each HLC is trained for approximately 1 million iterations, requiring about 7 days. All computations are performed on the CPU and no GPU-acceleration was leveraged.

7.1 LLC Performance

Figure 8 illustrates an LLC learning curve. Performance of intermediate policies are evaluated every 40k iterations by applying the policies for 32 episodes with a length of 20 s each. Performance is measured using the normalized cumulative reward (NCR), which is calculated as the sum of immediate rewards over an episode normalized by the minimum and maximum possible cumulative reward. No discounting is applied when calculating the NCR. A comparison between LLC's trained using 10 mocap clips, 1 hand-authored forward walking motion, and no reference motions, is also available in Figure 8. Since the LLC's use different reference motions, the NCR is measured using only the footstep terms $r = w_{end}r_{end} + w_{heading}r_{heading}$. A richer repertoire of reference motions leads to noticeable improvements in learning speed and final performance. Without a reference motion, the LLC fails to learn a successful walk. Learning curves for the stylized LLC's are available in the supplemental material.

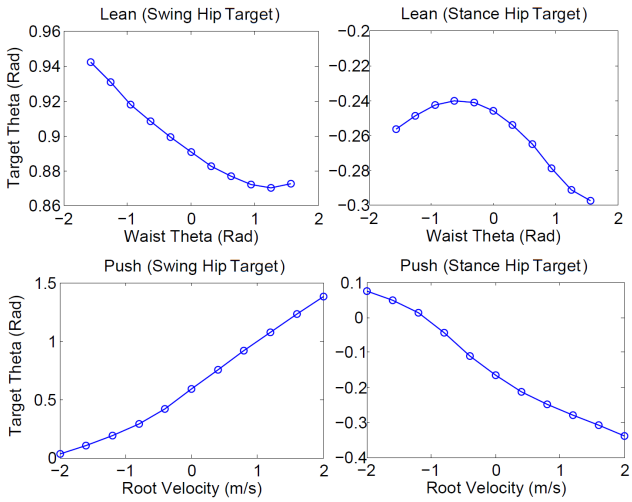

Fig. 9. PD target angles for the swing and stance hip as a function of character state. **top:** character's waist is leaning forward at various angles, with positive theta indicates a backward lean. **bottom:** the root is given a push at different velocities.

LLC Robustness: LLC's trained for different styles are evaluated for robustness by measuring the maximum perturbation force that each LLC can tolerate before falling. The character is first directed to walk forward, then a push is applied to the torso mid-point. Forward and sideways pushes were tested separately where each perturbation is applied for 0.25 s. The magnitude of the forces are increased in increments of 10 N until the character falls. We also evaluated the

Task	Reward (NCR)
Path Following	0.55
Soccer Dribbling	0.77
Pillar Obstacles	0.56
Block Obstacles	0.70
Dynamic Obstacles	0.18

Table 2. Performance summary of HLC's trained for each task. The NCR is calculated using the average of 256 episodes.

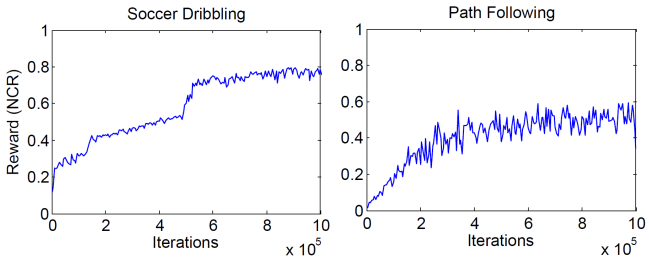

Fig. 10. HLC learning curves. Additional learning curves are available in the supplemental material.

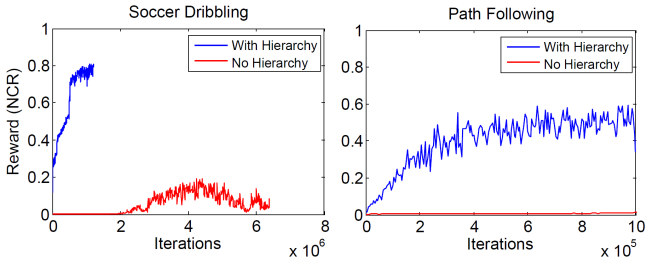

Fig. 11. Learning curves with and without control hierarchy.

LLC's robustness to terrain variation by measuring the steepest incline and decline that each LLC can successfully travel across without falling for 20 *s*. The maximum force that each LLC is able to recover from, and the steepest incline and decline are summarized in Table 1. The nominal walk proves fairly robust to the different perturbations, while the straight leg walks are generally less robust than the other styles. Though the LLC's were trained exclusive on flat terrain, the nominal LLC is able to walk up 16% inclines without falling. After normalizing for character weight and size differences, the robustness of the nominal walk LLC is comparable to figures reported for SIMBICON, which leverages manually-crafted balance strategies [Yin et al. 2007]. The LLC's robustness likely stems from the application of exploration noise during training. The added noise perturbs the character away from its nominal trajectory, requiring it to learn recovery strategies for unexpected perturbations. We believe that robustness could be further improved by presenting the character with examples of different pushes and terrain variations during training, and by letting it anticipate pushes and upcoming terrain. We also test for robustness with respect to changes in the gait period, i.e., forcing the controller to walk with shorter or longer duration steps. The gaits are generally robust to changes in gait period of ±20%.

To better understand the feedback strategies developed by the networks we analyze the action outputs from the nominal walk LLC for different character state configurations. Figure 9 illustrates the swing and stance hip target angles as a function of character's state. The state variations we consider include the waist leaning forward and backward at different angles, and pushing the root at different velocities. The LLC exhibits intuitive feedback strategies reminiscent of SIMBICON [Yin et al. 2007]. When the character is leaning too far forward or its forward velocity is too high, then the swing hip is raised higher to help position the swing foot further in front to regain balance in the following step, and vice-versa. but unlike SIMBICON, whose linear balance strategies are manually-crafted, the LLC develops nonlinear strategies without explicit user intervention.

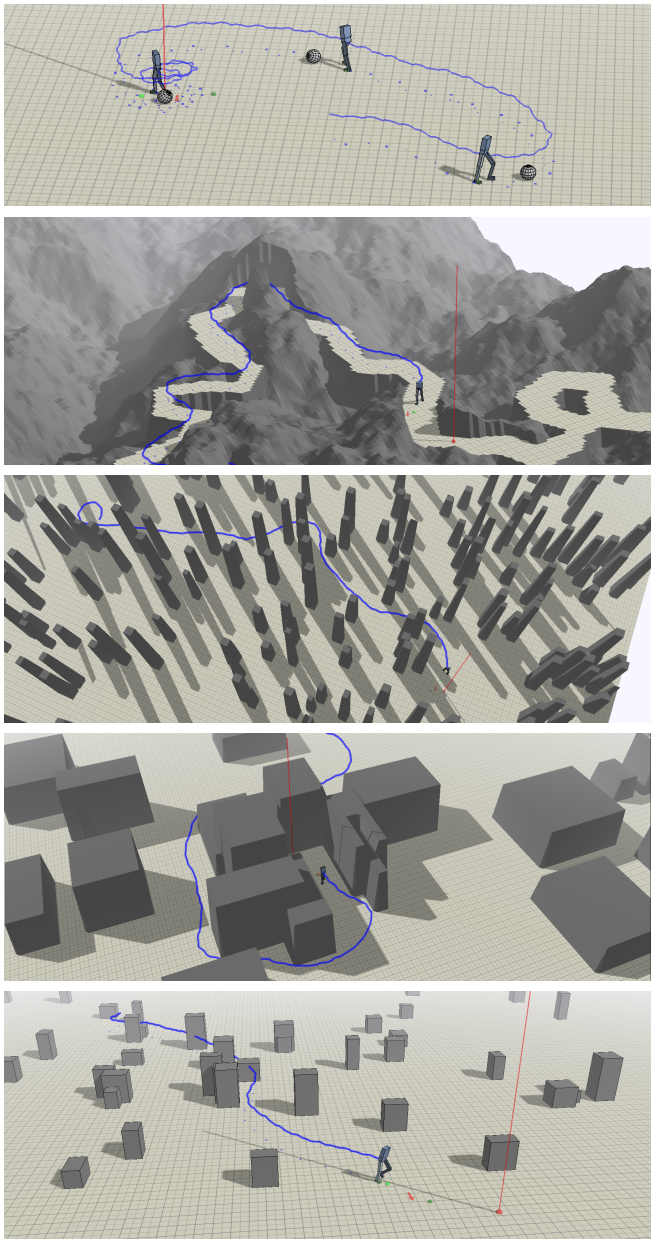

Fig. 12. Snapshots of HLC tasks. The red marker represents the target location and the blue line traces the trajectory of the character's center of mass. **top-to-bottom:** soccer dribbling, path following, pillar obstacles, block obstacles, dynamic obstacles.

7.2 HLC Performance

Learning curves for HLC's trained for different tasks are available in Figure 10. Intermediate policy performance is evaluated every 5k iterations using 32 episodes with a length of 200 s each. Note that the maximum normalized cumulative reward, NCR = 1, may not always be attainable. For soccer dribbling, the maximum NCR would require instantly moving the ball to the target location. For the navigation tasks, the maximum NCR would require a straight and unobstructed path between the character and target location.

For soccer dribbling, the HLC learns to correctly sequence the required sub-tasks. The HLC first directs the character towards the ball. It then dribbles the ball towards the target. Once the ball is sufficiently close to the target, the HLC developed a strategy of circling around the ball, while maintaining some distance, to avoid perturbing the ball away from the target or tripping over the ball. Alternatively, the ball can be replaced with a box, and the HLC is able to generalize to the different dynamics without additional training. The HLC's for the path following, pillar obstacles, and block obstacles tasks all learned to identify and avoid obstacles using heightmaps and navigate across different environments seeking randomly placed targets. For the path following task, the reward does not explicitly penalize the character for straying off the path, but the policy learns that falls are less likely when staying on the path. Therefore the path following behaviour emerges from the learning process. The more difficult dynamic obstacles environment, proved challenging for the HLC, reaching a competent level of performance, but still prone to occasional missteps, particularly when navigating around faster moving obstacles. We note that the default LLC training consists of constant speed forward walks and turns but no stopping, which limits the options available to the HLC when avoiding obstacles.

Figure 11 compares the learning curves with and without the control hierarchy for soccer dribbling and path following. To train the policies without the control hierarchy, the LLC's inputs are augmented with g_H and for the path following task, the terrain map T is also included as part of the input. Convolutional layers are added to the path following LLC. The augmented LLC's are then trained to imitate the reference motions and perform the high-level tasks. Without the hierarchical decomposition, both LLC's failed to perform their respective tasks.

7.3 Transfer Learning

Another advantage of a hierarchical structure is that it enables a degree of interchangeability between the different components. While a common LLC can be used by the various task-specific HLC's, a common HLC can also be applied to multiple LLC's without additional training. This form of zero-shot transfer allows the character to swap between different LLC's while retaining a reasonable level of aptitude for a task. Furthermore, the HLC can then be fine-tuned to improve performance with a new LLC, greatly decreasing the training time required when compared to retraining from scratch. In Figure 13 the performance when using different LLC's is shown for soccer dribbling before and after HLC fine-tuning, and retraining from scratch. Learning curves are available in Figure 14. Fine-tuning is applied for 200k iterations using the HLC trained for the nominal

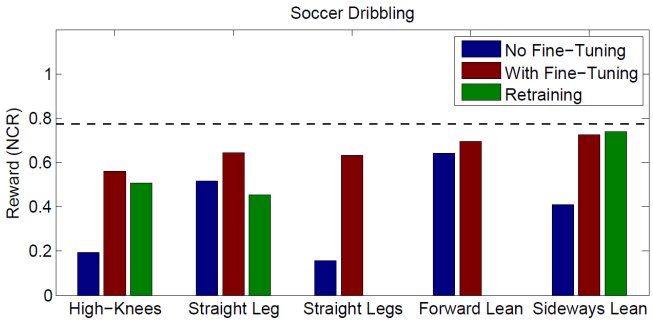

Fig. 13. Performance using different LLC's for soccer dribbling with and without HLC fine-tuning, and retraining.

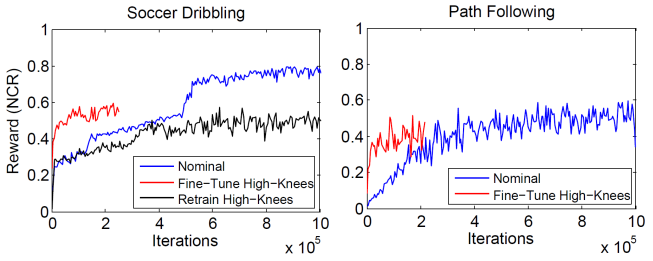

Fig. 14. Learning curves for fine-tuning HLC's trained for the nominal LLC to different LLC's, and retraining HLC's from scratch.

LLC for initialization. Retraining is performed for 1 million iterations from random initialization. For soccer dribbling, the ability to substitute different LLC's is style dependent, with the forward lean exhibiting the least degradation and high-knees exhibiting the most.

8 DISCUSSION

The method described in this paper allows for skills to be designed while making few assumptions about the controller structure or explicit knowledge of the underlying dynamics. Skill development is guided by the use of objective functions for low-level and high-level policies. Taken together, the hierarchical controller allows for combined planning and physics-based movement based on high-dimensional inputs. Overall, the method further opens the door to learning-based approaches that allow for rapid and flexible development of movement skills, at multiple levels of abstraction. The same deep RL method is used at both timescales, albeit with different states, actions, and rewards. Taken as a whole, the method allows for learning skills that directly exploit a variety of information, such as the terrain maps for navigation-based tasks, as well as skills that require finer-scale local interaction with the environment, such as soccer dribbling.

Imitation objective: The LLC learns in part on a motion imitation objective, utilizing a reference motion that provides a sketch of the expected motion. This can be as simple as a single keyframed planar walk cycle that helps guide the control policy towards a reasonable, readily-available movement pattern, as opposed to learning it

completely from scratch. Importantly, it further a means of directing the desired motion style, given the lack of detailed biological modeling that would otherwise likely be needed to achieve natural motions purely as the product of optimization. While a single crude keyframed planar walk cycle can produce good LLC policies that also learn to turn, the use of multiple motion capture clips enables sharper and more natural turns. Once a basic control policy is in place, the policy can be further adapted using new goals or objective functions, as demonstrated in our multiple LLC style variations.

Phase information: Our LLC's currently still use phase information as part of the character state, which can be seen as a basic memory element, i.e., "where am I in the gait cycle." We still do not fully understand why a bilinear phase representation works better for LLC learning, in terms of achieving a given motion quality, than the alternative of using continuously-valued phase representation, i.e., $\cos(\phi)$, $\sin(\phi)$. In future work, we expect that the phase could be stored and updated using an internal memory element in a recurrent network or LSTM. This would also allow for phase adaptations, such as stopping or reversing the phase advancement when receiving a strong backwards push.

HLC-LLC interface and integration: Currently, the representation used as the interface between the HLC and LLC, $a_H \equiv g_L$, is manually specified, in our case corresponding to the next two footstep locations and the body orientation at the first footstep. The HLC treats these as abstract handles for guiding the LLC, and thus may exploit regions of this domain for which the LLC has never been trained. This is evident in the HLC behaviour visualizations, which show that unattainable footsteps are regularly demanded of the LLC by the HLC. This is not problematic in practice because the HLC will learn to avoid regions of the action space that lead to problematic behaviors from the LLC, such as falling. Learning a suitable representation for the interface between the HLC and LLC, as demonstrated in part by [Heess et al. 2016], is an exciting avenue for future work. It may be possible to find representations which then allow for LLC substitutions with less performance degradation. An advantage of the current explicitly-defined LLC goals, g_L, is that it can serve to define the reward to be used for LLC training. However, it does result in the LLC's and HLC's being trained using different reward functions, whereas a more conceptually pure approach might simply use a single objective function.

Motion planning: Some tasks, such as path navigation, could also be accomplished using existing motion planning techniques based on geometric constraints and geometric objectives. However, developing efficient planning algorithms for tasks involving dynamic worlds, such as the dynamic obstacles task or the soccer dribbling task, is much less obvious. In the future, we also wish to develop skills that are capable of automatically selecting efficient and feasible locomotion paths through challenging 3D terrains.

Transfer and parameterization: Locomotion can be seen as encompassing a parameterized family of related movements and skills. Knowing one style of low-level motion should help in learning another style, and, similarly, knowing the high-level control for one task, e.g., avoiding static obstacles, should help in learning another related task, e.g., avoiding dynamic obstacles. This paper has demonstrated several aspects of transfer and parameterization. The ability to interpolate (and to do moderate extrapolation) between different LLC motion styles provides a rich and conveniently parameterized space of motions. The LLC motions are robust to moderate terrain variations, external forces, and changes in gait period, by virtue of the exploration noise they experience during the learning process. As demonstrated, the HLC-LLC hierarchy also allows for substitution of HLC's and LLC's. However, for HLC/ LLC pairs that have never been trained together, the performance will be degraded for tasks that are sensitive to the dynamics, such as soccer dribbling. However, the HLC's can be efficiently readapted to improve performance with additional fine-tuning.

Learning efficiency: The sample efficiency of the training process can likely be greatly improved. Interleaving improvements to a learned dynamics model with policy improvements is one possible approach. While we currently use a positive-temporal difference advantage function in our actor-critic framework, we intend to more fully investigate other alternatives in future work.

End-to-end training: While the HLC and LLC are currently trained separately, we have experimented with training all levels of the hierarchy simultaneously. However, this led to a number of challenges. In particularly, the dynamics observed by the HLC is determined by the behaviour of the LLC. When both are trained jointly, the changing LLC dynamics also impacts the HLC and impairs its ability to learn. Exploring methods for end-to-end training of hierarchical policies is an exciting direction for future work.

9 CONCLUSIONS

We have presented a hierarchical learning-based framework for 3D bipedal walking skills that makes limited use of prior structure. It allows for easily-directable control over the motion style and is shown to produce highly robust controllers. The controllers can directly exploit terrain maps and high-dimensional state descriptors for the character. The hierarchical decomposition allows for both high-level and low-level controllers to be reused.

There are many exciting directions to explore. We wish to explore how further agility can be achieved, for walking and running. We would like to explore how to best learn to walk and run directly over 3D terrains, whereas our current control tasks navigate on flat paths through 3D terrains. It will be important in the future to find ways of learning additional LLC skills that are directly in support of a given HLC task. An interesting direction would be to learn adversarial and cooperative behaviors between multiple characters, such as for playing tag or for collision avoidance when crossing paths. In the future, we also hope to be able to directly learn to drive muscle-actuated 3D models.

REFERENCES

Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *TVCG* 19, 8 (2013), 1405–1414.

Yunfei Bai, Kristin Siu, and C Karen Liu. 2012. Synthesis of concurrent object manipulation tasks. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 156.

Bullet. 2015. Bullet Physics Library. (2015). http://bulletphysics.org.

Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade. 2005. Footstep Planning for the Honda ASIMO Humanoid. In *ICRA05*. 629–634.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust task-based control policies for physics-based characters. *ACM Transctions on Graphics* 28, 5 (2009), Article 170.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Transctions on Graphics* 29, 4 (2010), Article 130.

Stelian Coros, Philippe Beaudoin, Kang Kang Yin, and Michiel van de Panne. 2008. Synthesis of constrained walking skills. *ACM Trans. Graph.* 27, 5 (2008), Article 113.

Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. 2011. Locomotion Skills for Simulated Quadrupeds. *ACM Transactions on Graphics* 30, 4 (2011), Article 59.

Marco da Silva, Yeuhi Abe, and Jovan Popović. 2008. Interactive simulation of stylized human locomotion. *ACM Trans. Graph.* 27, 3 (2008), Article 82.

Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. 2010. Feature-based locomotion controllers. In *ACM Transactions on Graphics (TOG)*, Vol. 29. ACM, 131.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding. *CoRR* abs/1606.01847 (2016). http://arxiv.org/abs/1606.01847

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 2492–2515.

Michael X. Grey, Aaron D. Ames, and C. Karen Liu. 2016. Footstep and Motion Planning in Semi-unstructured Environments Using Possibility Graphs. *CoRR* abs/1610.00700 (2016). http://arxiv.org/abs/1610.00700

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 9–20.

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

Nikolaus Hansen. 2006. The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation*. 75–102.

Nicolas Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. 2016. Learning and Transfer of Modulated Locomotor Controllers. *CoRR* abs/1610.05182 (2016). http://arxiv.org/abs/1610.05182

J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. 1995. Animating Human Athletics. In *Proceedings of SIGGRAPH 1995*. 71–78.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia (MM '14)*. ACM, 675–678. DOI: https://doi.org/10.1145/2647868.2654889

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics & Automation* 12, 4 (1996), 566–580.

James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. 2005. *Motion Planning for Humanoid Robots*. Springer Berlin Heidelberg, 365–374.

Manfred Lau and James Kuffner. 2005. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 271–280.

Jehee Lee and Kang Hoon Lee. 2004. Precomputing Avatar Behavior from Human Motion Data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '04)*. 79–87.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-Driven Biped Control. *ACM Transctions on Graphics* 29, 4 (2010), Article 129.

Sergey Levine and Pieter Abbeel. 2014. Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger (Eds.). Curran Associates, Inc., 1071–1079.

Sergey Levine and Vladlen Koltun. 2013. Guided Policy Search. In *ICML '13: Proceedings of the 30th International Conference on Machine Learning*.

Sergey Levine and Vladlen Koltun. 2014. Learning complex neural network policies with trajectory optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 829–837.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

Libin Liu, Michiel van de Panne, and KangKang Yin. 2016. Guided Learning of Control Graphs for Physics-based Characters. *ACM Trans. Graph.* 35, 3 (2016), Article 29. DOI: https://doi.org/10.1145/2893476

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

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum Control for Balance. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. ACM, New York, NY, USA, Article 80, 8 pages. DOI: https://doi.org/10.1145/1576246.1531386

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783 (2016). http://arxiv.org/abs/1602.01783

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.* 29, 4 (2010), Article 71.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. 2015. Interactive Control of Diverse Complex Characters with Neural Networks. In *Advances in Neural Information Processing Systems*. 3114–3122.

Igor Mordatch and Emanuel Todorov. 2014. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*.

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.* 28, 3 (2009), Article 81.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, Johannes FÃijrnkranz and Thorsten Joachims (Eds.). Omnipress, 807–814. http://www.icml2010.org/papers/432.pdf

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Transactions on Graphics* 34, 4 (2015), Article 80.

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Transactions on Graphics* 35, 4 (2016), Article 81.

Xue Bin Peng and Michiel van de Panne. 2016. Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter? *CoRR* abs/1611.01055 (2016). http://arxiv.org/abs/1611.01055

Ken Perlin. 2002. Improving Noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*. ACM, New York, NY, USA, 681–682. DOI: https://doi.org/10.1145/566570.566636

Julien PettrÃĺ, Jean-Paul Laumond, and Thierry SimÃĺon. 2003. 2-Stages Locomotion Planner for Digital Actors. In *SCA '03: Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 258–264.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust Region Policy Optimization. *CoRR* abs/1502.05477 (2015). http://arxiv.org/abs/1502.05477

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR 2016)*.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph.* 26, 3 (2007), Article 107.

Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *In Advances in Neural Information Processing Systems 12*. MIT Press, 1057–1063.

Jie Tan, Yuting Gu, C Karen Liu, and Greg Turk. 2014. Learning bicycle stunts. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 50.

Jie Tan, Karen Liu, and Greg Turk. 2011. Stable proportional-derivative controllers. *Computer Graphics and Applications, IEEE* 31, 4 (2011), 34–44.

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

Hado Van Hasselt. 2012. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*. Springer, 207–251.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2009. Optimizing Walking Controllers. *ACM Transctions on Graphics* 28, 5 (2009), Article 168.

David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howardy, Alfred A. Rizzi, and Marc Raibert. 2010. Autonomous Navigation for BigDog. In *ICRA10*. 4736–4741.

Jia-chi Wu and Zoran Popović. 2010. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics* 29, 4 (Jul. 2010), 72:1–72:10.

Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. 2004. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph.* 23, 3 (2004), 532–539.

Yuting Ye and C. Karen Liu. 2010. Optimal Feedback Control for Character Animation Using an Abstract Model. *ACM Trans. Graph.* 29, 4 (2010), Article 74.

KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2008. Continuation Methods for Adapting Simulated Skills. *ACM Transctions on Graphics* 27, 3 (2008), Article 81.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Transctions on Graphics* 26, 3 (2007), Article 105.

Petr Zaytsev, S Javad Hasaneini, and Andy Ruina. 2015. Two steps is enough: no need to plan far ahead for walking balance. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6295–6300.

SUPPLEMENTARY MATERIAL

10 LLC WALK CYCLES

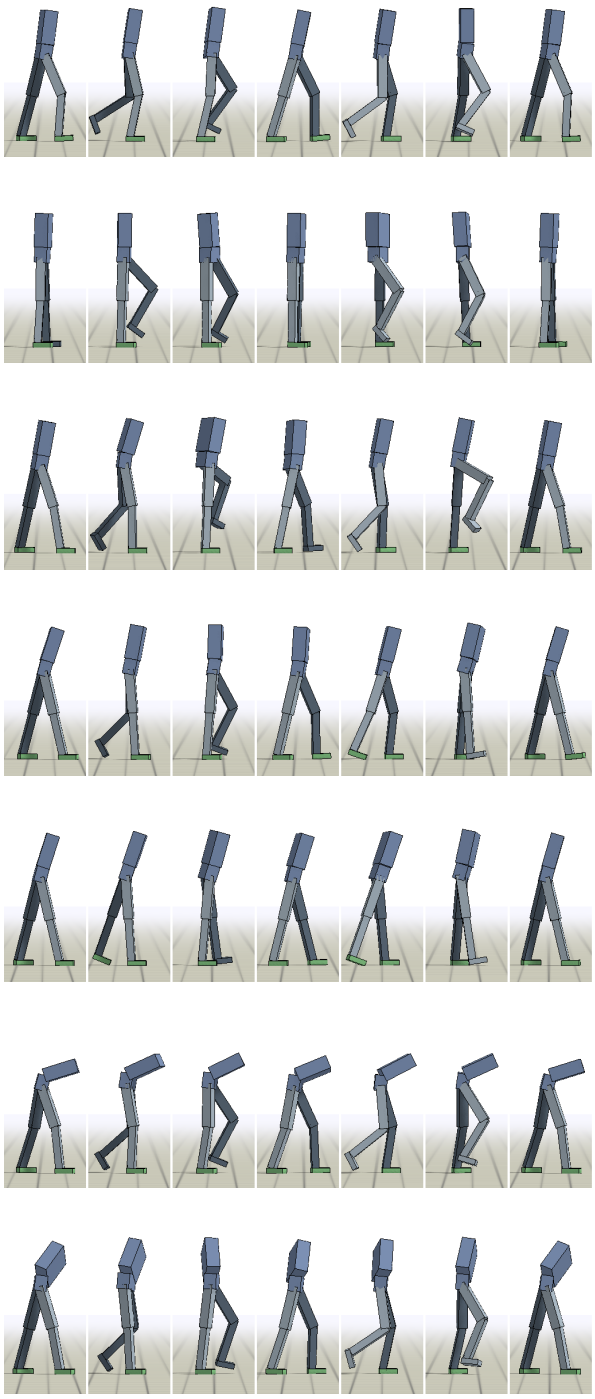

Fig. 15. LLC walk cycles. **top-to-bottom:** nominal walk, in-place walk, high-knees, straight leg, straight legs, forward lean, sideways lean.

11 LLC REWARD

$$r_L = w_{pose} r_{pose} + w_{vel} r_{vel} + w_{root} r_{root} + w_{com} r_{com} + w_{end} r_{end} + w_{heading} r_{heading}$$

$$r_{pose} = \exp\left(-\sum_i w_i d(\hat{q}_i(t), q_i)^2\right)$$

$$r_{vel} = \exp\left(-\sum_i w_i ||\hat{\dot{q}}_i(t) - \dot{q}_i||^2\right)$$

$$r_{root} = \exp\left(-10(\hat{h}_{root} - h_{root})^2\right)$$

$$r_{com} = \exp\left(-||\hat{v}_{com} - v_{com}||^2\right)$$

$$r_{end} = \exp\left(-||\hat{p}_{swing} - p_{swing}||^2 - ||\hat{p}_{stance} - p_{stance}||^2\right)$$

$$r_{heading} = 0.5\cos\left(\hat{\theta}_{root} - \theta_{root}\right) + 0.5$$

h_{root} represents the height of the root from the ground, v_{com} is the center of mass velocity, p_{swing} and p_{stance} are the positions of the swing and stance foot. The target position for the swing foot $\hat{p}_{swing} = \hat{p}_0$ is provided by the footstep plan, while the target position for the stance foot $\hat{p}_{stance}$ is provided by the reference motion. θ_{root} represents the heading of the root on the horizontal plane, and $\hat{\theta}_{root}$ is the desired heading provided by the footstep plan.

12 HLC SOCCER DRIBBLING REWARD

$$r_H = w_{cv} r_{cv} + w_{cp} r_{cp} + w_{bv} r_{bv} + w_{bp} r_{bp}$$

$$r_{cv} = \exp\left(-\left(\min(0, u_{ball}^T v_{com} - \hat{v}_{com})\right)^2\right)$$

$$r_{cp} = \exp\left(-d_{ball}^2\right)$$

$$r_{bv} = \exp\left(-\left(\min(0, u_{tar}^T v_{ball} - \hat{v}_{ball})\right)^2\right)$$

$$r_{bp} = \exp\left(-d_{tar}^2\right)$$

u_{ball} is a unit vector pointing in the direction from the character to the ball, v_{com} the character's center of mass velocity, and $\hat{v}_{com} = 1m/s$ the desired speed with which the character should move towards the ball. Similarly, u_{tar} represents the unit vector pointing from the ball to the target position, v_{ball} the velocity of the ball, and $\hat{v}_{ball} = 1m/s$ the desired speed for the ball with which to move towards the target. Once the ball is within 0.5 m of the target and the character is within 2 m of the ball, then the goal is considered fulfilled and the character receives a constant reward of 1 from all terms, corresponding to the maximum possible reward.

13 LLC STYLIZATION

Learning curves for each stylized LLC is available in Figure 16. Each network is initialized using the LLC trained for the nominal walk. Performance is measured using only the style term c_{style}, measuring the LLC's conformity to the style objectives.

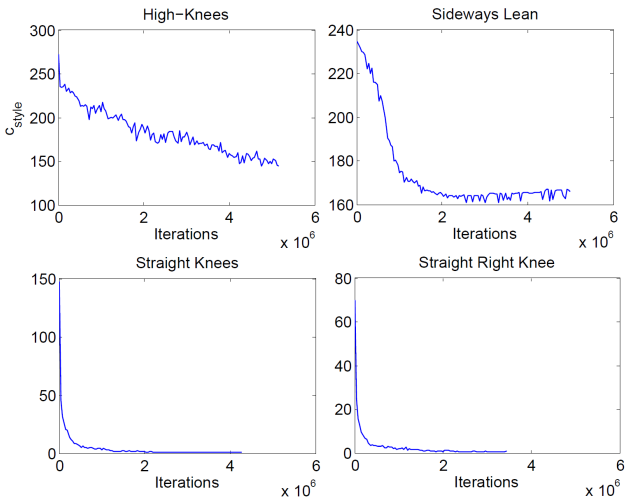

Fig. 16. Learning curves for each stylized LLC.

14 HLC LEARNING CURVES

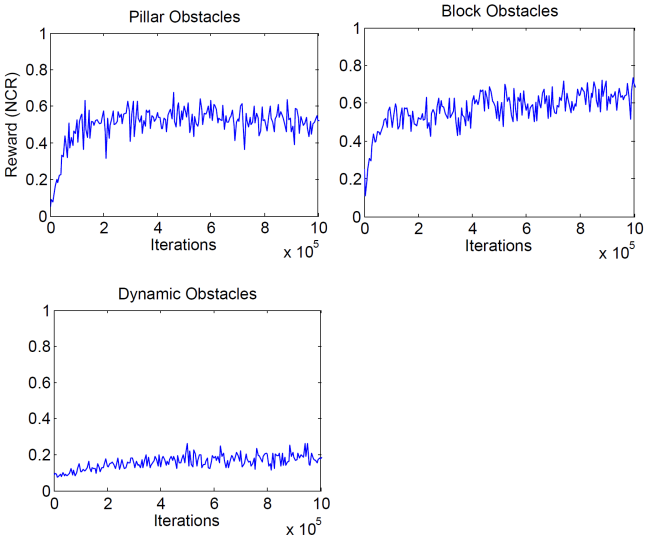

Fig. 17. HLC learning curves.

15 TRANSFER

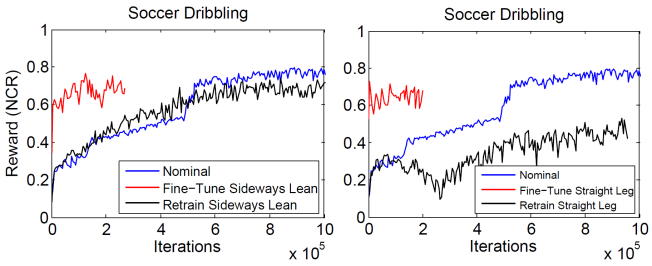

Fig. 18. Learning curves for fine-tuning HLC's for different LLC's, and retraining HLC's from scratch.

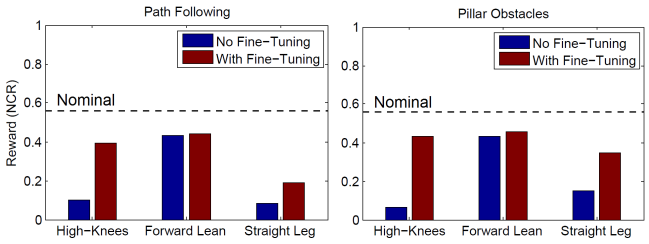

Fig. 19. Performance using different LLC's for path following and pillar obstacles with and without HLC fine-tuning.

Task + LLC	No Fine-Tuning	With Fine-Tuning	Retraining
Soccer + High-Knees	0.19	0.56	0.51
Soccer + Straight Leg	0.51	0.64	0.45
Soccer + Straight Legs	0.16	0.63	-
Soccer + Forward Lean	0.64	0.69	-
Soccer + Sideways Lean	0.41	0.72	0.74
Path + High-Knees	0.10	0.39	-
Path + Forward Lean	0.43	0.44	-
Path + Straight Leg	0.08	0.19	-
Pillars + High-Knees	0.06	0.43	-
Pillars + Forward Lean	0.43	0.45	-
Pillars + Straight Leg	0.15	0.35	-

Table 3. Performance (NCR) of different combinations of LLC's and HLC's. **No Fine-Tuning:** directly using the HLC's trained for the nominal LLC. **With Fine-Tuning:** HLC's fine-tuned using the nominal HLC's as initialization. **Retraining:** HLC's are retrained from random initialization for each task and LLC.