Learning Task-Agnostic Action Spaces for Movement Optimization

Amin Babadi, Michiel van de Panne, C. Karen Liu, and Perttu Hämäläinen

Abstract—We propose a novel method for exploring the dynamics of physically based animated characters, and learning a task-agnostic action space that makes movement optimization easier. Like several previous article, we parameterize actions as target states, and learn a short-horizon goal-conditioned low-level control policy that drives the agent's state towards the targets. Our novel contribution is that with our exploration data, we are able to learn the low-level policy in a generic manner and without any reference movement data. Trained once for each agent or simulation environment, the policy improves the efficiency of optimizing both trajectories and high-level policies across multiple tasks and optimization algorithms. We also contribute novel visualizations that show how using target states as actions makes optimized trajectories more robust to disturbances; this manifests as wider optima that are easy to find. Due to its simplicity and generality, our proposed approach should provide a building block that can improve a large variety of movement optimization methods and applications.

Index Terms—Movement optimization, trajectory optimization, policy optimization, hierarchical reinforcement learning, action space

1 Introduction

Movement optimization of physically simulated characters is common in robotics and computer animation. Although previous work has shown that a wide variety of movements can be generated through optimization, both in trajectory and policy optimization settings [1], [2], [3], [4], [5], optimization is still prohibitively slow for many applications. This is largely due to the high dimensionality of the state and action spaces, as well as the complexity of movement dynamics plagued with contact discontinuities, which prohibits closed-form solutions.

One way to make movement optimization easier is by using novel action parameterizations. Previous work has shown that the choice of action space can have a significant impact on the performance of movement optimization [6]. Learned or designed action spaces can also capture the task-invariant behavioral representations that can be reused across different tasks, speeding up the problem solving process [7]. This idea has been long studied in the context of Hierarchical Reinforcement Learning (HRL), which has shown promising results in physically based character control [8], [9], [10]. In HRL, the actions output by a high-level controller (HLC) are converted into low-level simulation or robot actuation commands using a low-level controller (LLC). For example, the muscle-actuated human simulation system of Lee *et al*. [10] optimizes controls in the space of joint target accelerations, which are then converted to muscle actuations by a separately trained neural network. However, learning such control hierarchies can be computationally expensive and they are typically problem-dependent. A question remains whether there are *truly generic action parameterizations or LLCs that would only need to be trained once*, but would still make movement optimization and learning easier across a wide range of tasks.

In this paper, we investigate a promising candidate for a generic task-agnostic action space: *We define HLC actions as target states to be reached by an LLC*. This poses no limitations on the optimized movements, as any movement can be described as a sequence of state variables such as body poses and root translations and rotations. This can also be motivated through human visuomotor control and movement pedagogy, as complex movement skills such as gymnastics are typically taught through demonstrations—visualizations of target state sequences—instead of explaining movements through low-level actions such as which muscles to contract and when. Hence, reaching and maintaining desired state variable values can be considered a central human meta-learning or "learning to learn" skill. Furthermore, recent research indicates that parameterizing actions as target states can improve both convexity and conditioning of movement optimization [11]. However, the analysis of [11] was limited to an inverted pendulum, in which case a simple P-controller suffices as the LLC. Generalization of the results to neural network LLCs and more complex agents was not demonstrated, calling for further research.

This paper makes the following contributions:

- We propose a novel random exploration scheme for generating highly diverse training data for task-agnostic state-reaching LLCs, without dependencies on pre-recorded reference data. This makes our

- *Amin Babadi and Perttu Hämäläinen are with the Department of Computer Science, Aalto University, 02150 Espoo, Finland. E-mail: {amin.babadi, perttu.hamalainen}@aalto.fi.*
- *Michiel van de Panne is with the Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, Canada. E-mail: van@cs.ubc.ca.*
- *C. Karen Liu is with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA. E-mail: karenliu@cs.stanford.edu.*

Manuscript received 22 Sept. 2020; revised 30 June 2021; accepted 17 July 2021. Date of publication 27 July 2021; date of current version 27 Oct. 2022. (Corresponding author: Amin Babadi.)
Recommended for acceptance by J. Barbic.
Digital Object Identifier no. 10.1109/TVCG.2021.3100095

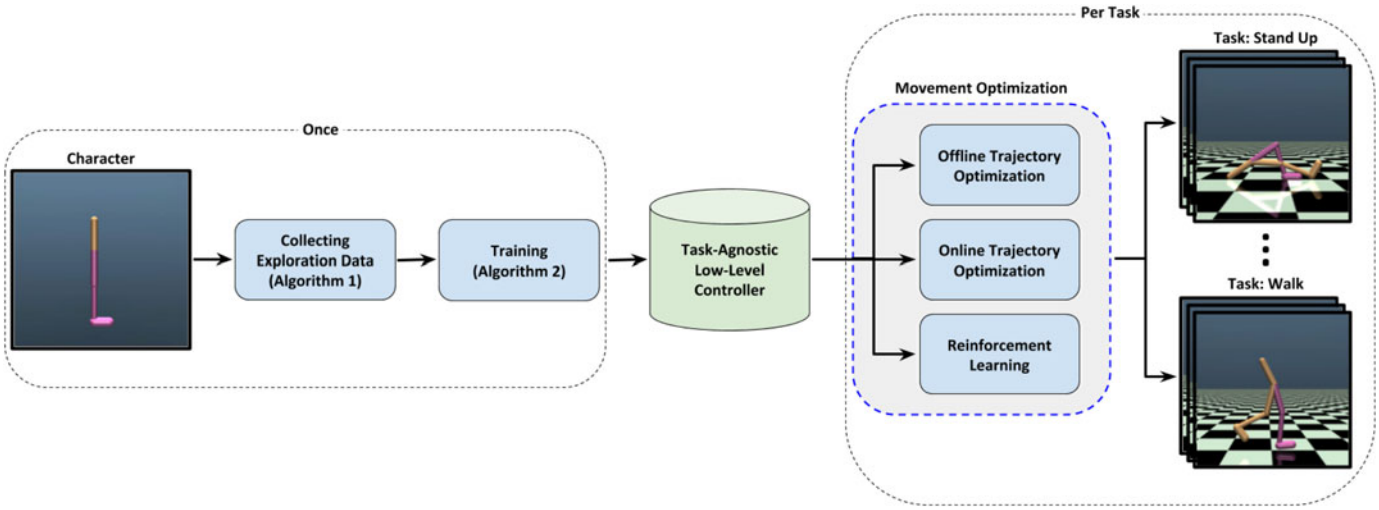


Fig. 1. The system pipeline consists of three main steps: 1) collecting random exploration data to discover feasible states and actions of the simulation environment; 2) using the exploration data to train low-level controller; and 3) movement optimization using either offline/online trajectory optimization or reinforcement learning. The first two steps are task-agnostic and only need to be completed once for each environment.

approach suitable for a wide variety of simulated characters and movements.

- Through extensive simulation experiments, we show that our LLCs improve movement optimization across multiple complex agents and multiple optimization methods including offline trajectory optimization, online trajectory optimization, and reinforcement learning (policy optimization). Our data also indicates that LLCs trained with the proposed exploration approach perform better than LLCs trained with simple random exploration data.
- We visualize the resulting optimization landscapes with and without the LLC, extending the investigation of [11] to complex agents and learned LLCs. Our results provide support for the improved convexity and conditioning suggested by [11].

An overview of our system and approach is shown in Fig. 1. To augment the quantitative data presented in Section 7, examples of the exploration behaviors and movement optimization results are included in the supplemental video[1]. To the best of our knowledge, no previous work has conducted such comprehensive experiments on a task-agnostic action space that helps both trajectory and policy optimization. Our results indicate that an LLC like ours should provide a useful basis for building any system that utilizes movement optimization. All of the implementations used in this work can be found at https://github.com/donamin/llc.

2 Related Work

Below, we review relevant previous work in the related areas of action space engineering, trajectory optimization, reinforcement learning, and movement exploration.

2.1 Action Space Engineering

One of the earliest successful examples of efficient action spaces is the Simple Biped Locomotion Control (SIMBICON) [12]. The control strategy of SIMBICON includes a Finite State Machine (FSM) whose states correspond to different phases of a biped walking cycle. This controller is able to produce robust locomotion movements using a small number of parameters, which can be tuned either manually or using motion capture data. This action parameterization has shown to be expressive enough to synthesize novel movements through interpolation and extrapolation [13]. SIMBICON's control strategy has also been extended to muscle-based control settings to synthesize high-quality locomotion gaits for bipedal creatures [14]. The latter uses Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [15] to optimize the controller parameters.

2.2 Trajectory Optimization

Parameterizing action sequences as splines is a modified action space common in synthesizing physically-based movements through trajectory optimization, i.e., optimizing actions to maximize or minimize an objective function that encodes movement goals as a function of both the action sequence and the resulting state-space movement trajectory. A spline parameterization reduces the problem dimensionality by expressing a long action sequence using only a few control points and interpolating between them. Plus, it enforces coordination across body joints (similar to SIMBICON [12]), which leads to more natural and smooth movements.

In offline settings, spline parameterization has been used in Contact-Invariant Optimization (CIO), a method that simultaneously optimizes the contact and behavior in different phases of the movement [16]. In another work, it has been used for building a low-level controller that uses CMA-ES for synthesizing humanoid wall climbing movements [17].

In online settings, splines and sequential Monte Carlo sampling have been used for synthesizing interactive humanoid movements [18]. In a similar work, CMA-ES empowered by two seeding techniques was used to generate interactive martial arts movements for upper-body humanoid characters [19].

1. https://youtu.be/s8VARNpBpSg

2.3 Reinforcement Learning

Finding novel action spaces is more widely studied in reinforcement learning than in trajectory optimization. This dates back at least to the early 1990s with Feudal Reinforcement Learning, that proposed solving the RL problem in multiple resolutions simultaneously [20]. The idea was then extended to temporally extended sequence of actions, also known as the Options framework [21]. The more general form is now known as Hierarchical Reinforcement Learning (HRL) [22]. In HRL, a Low-Level Controller (LLC) is used for handling atomic actions, and policy optimization is applied to produce a High-Level Controller (HLC) that controls the agent via interaction with the LLC. The LLC can be either designed or built using reinforcement learning or other optimization methods.

Inspired by SIMBICON [12], phase-based FSMs are one of the most popular action parameterizations in reinforcement learning. They have been used to learn dynamic terrain traversal policies for various 2D characters [23]. The same approach has been extended to Mixture of Actor-Critic Experts (MACE) to increase the training speed and enable expert specialization [24].

Another HRL approach is to define HLC as a gating network whose job is to choose among a number of low-level primitive controllers. In this setting, each primitive controller is specialized in performing a specific behavior (e.g., walking forward) or controlling a specific subset of joints (e.g., upper body). This approach has been used in [25], where Deep Q-Learning [26] is used to learn a scheduler (i.e., HLC) that chooses one of the many control fragments (i.e., LLCs) to control the character for the next 0.1 seconds. Another similar work, called Multiplicative Compositional Policies (MCP), first trains several LLCs to imitate several short motion capture clips, and then trains a HLC to, at each timestep, compute a weighted composition of the LLCs' outputs [27].

In some HRL studies, the HLC is used to specify the long-term high-level goals that the LLC needs to achieve. A good example in this category is called DeepLoco, where the HLC is responsible for planning next footsteps and the LLC applies necessary low-level actions for satisfying the plan [8]. Another work has used end-to-end representation learning to find near-optimal goal spaces for continuous tasks [28].

A popular HRL approach is to use a LLC that approximates inverse dynamics, i.e., outputs low-level actions required to take the agent to some desired state. Hindsight Experience Replay (HER) [29] and its recent hierarchical extension [9] use this idea by augmenting the experience replay dataset such that all reachable states can be considered as potential goal states. Another example is HIRO, a method that trains both HLC and LLC concurrently using off-policy experiences [30].

Several studies use reference animations to train robust control policies that demonstrate natural movement. One study uses adversarial imitation learning to train the LLCs, which are then used in RL settings to produce human-like motions [31]. A similar approach is used to learn control policies for quadruped characters [32]. In another study, LLC policies are trained to replicate short movement primitives of length 0.1 to 0.3 seconds, and then a HLC is trained to select among those LLCs based on visual input [33]. In a similar study, a single LLC is built by training an autoencoder on multiple expert policies [34]. Another study trains general control policy by compressing a large number of expert policies that replicate motion capture moves [35]. A recent work uses a large corpora of motion capture clips to learn a low-level stochastic embedding space, which is then used in different tasks [36]. A recent stream of research trains a kinematic motion generator on top of a state-reaching LLC to synthesize high-quality humanoid animation [3], [5], [37]. However, these systems need motion capture data for training and the synthesized movements are goal-conditioned variations of the data. In contrast, our focus is on training the LLCs and optimizing movement without reference data, using a simple but efficient and task-agnostic exploration method. At least in principle, this should impose less limitations on the creativity and diversity of the results.

It should be noted that reinforcement learning and trajectory optimization can also be combined. For example, [38], [39], [40] use trajectory optimization to guide the training of a neural network policy. Trajectory optimization can also be used to generate states and actions for initializing an RL algorithm [41] or to synthesize reference movement trajectories for imitation learning using RL [42].

2.4 Exploration

Using exploration to handle the uncertainty is one of the building blocks of optimal control [43]. In order to train a goal-conditioned low-level controller, one needs a proper dataset that includes the movements of interest and covers relevant regions of the state space. In the absence of a supervised dataset, the only way is to use some exploration approach to create one. Such random exploration is common in model-based reinforcement learning, where the exploration data is used to learn a forward dynamics model [44], [45], [46].

Exploration is typically formulated as an intrinsically motivated optimization/learning problem, with a reward function that simulates the psychological mechanisms driving learning and exploration in biological agents, e.g., a drive to seek and experience novel or unpredictable stimuli. One implemention of this is to maximize the undiscounted information gain [47]. Another example is called TEXPLORE-VANIR, a method whose reward encourages exploration when the model is uncertain or a novel experience is possible [48]. A pseudo-count model based on density estimation has also been proposed to measure the novelty of encountered states in Atari 2600 environments [49]. In a more recent work called Dreamer, exploration is done through imagination using a learned world model of latent state space [50]. In this paper, we propose a simple random exploration approach that does not need novelty or density estimates, but still achieves much better coverage of possible agent states and produces more capable LLCs than naive exploration with random actions.

Recently, Sekar *et al.* [51] have also conducted experiments in improving movement exploration, but using an approach largely orthogonal to ours. They proposed to train a world model using data produced through maximizing

the disagreement among an ensemble of one-step forward models. Such forward models map an observation and action to the next observation, allowing model-based or "imagination-based" planning. In contrast, we learn inverse models that allow control optimization to operate in the space of the next observation(s). In addition, Sekar *et al*. [51] focused on reinforcement learning and relatively simple agents like 2D HalfCheetah and Pendulum, whereas we investigate both RL and trajectory optimization and include a 3D humanoid model in our experiments.

3 Preliminaries

Before going into the details of our approach, the following reviews the optimization methods and problem definitions we utilize.

3.1 Trajectory Optimization

Trajectory optimization denotes the process of searching for an optimal sequence of actions, which produce some desired movement trajectory when applied to a dynamical system. With differentiable dynamics, trajectory optimization can utilize gradient information [1], which is however unreliable with complex contact discontinuities. This can be overcome by sampling-based trajectory optimization [18], [52] which works with any black-box dynamics simulator. The latter is the setting explored in this paper.

The core of sampling-based trajectory optimization consists of a simple iteration loop: 1) sample a number of random action trajectories and use them to simulate the state trajectories, 2) evaluate them using some cost or reward function, 3) use the best scoring trajectories to refine the sampling distribution for the next iteration. Finally, the best found trajectory is picked as the solution.

Trajectory optimization can be done offline or online. In the offline case, the starting state is fixed and the target is to find a single trajectory to achieve some goal. In online optimization case, also known as Model Predictive Control (MPC), when a trajectory is found, the simulation progresses by one timestep and the optimizer is asked to generate another trajectory starting from the next timestep. In other words, only the first action in the trajectory is actually executed and the rest is only used to compute the cost function and updating the sampling distribution.

We simulate each sampled trajectory until a time horizon T and the cost function is defined as the sum of costs over all timesteps. Suppose the character's state at the beginning of optimization is $\mathbf{s_t} \in \mathcal{S}$ ($t = 0$ in the case of offline optimization). Forward simulating a sequence of actions $\{\mathbf{a_t}, \mathbf{a_{t+1}}, \ldots, \mathbf{a_{t+T-1}}\} \in \mathcal{A}^T$ results in a trajectory $(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}, \mathbf{a_{t+1}}, \ldots, \mathbf{s_{t+T-1}}, \mathbf{a_{t+T-1}}, \mathbf{s_{t+T}})$. Then, the problem will be to find the action sequence that minimizes the following accumulative cost:

$$\sum_{i=t}^{t+T-1} [C_{\mathcal{A}}(\mathbf{a_i}) + C_{\mathcal{S}}(\mathbf{s_{i+1}})],$$

where $C_{\mathcal{A}}$ and $C_{\mathcal{S}}$ are functions for computing the action and state costs, respectively. $C_{\mathcal{S}}$ usually encodes some information about the target movement. For example, if the goal is to produce walking movement, $C_{\mathcal{S}}$ can penalize the difference between the current and desired velocities, and $C_{\mathcal{A}}$ can penalize the amount of torques used in the simulation. This setup will encourage the character to move with the desired velocity while avoiding extreme movements [53].

3.2 Reinforcement Learning

In reinforcement learning, the learning process involves an agent interacting with an environment by observing a state, applying an action, and receiving a reward. The goal is to repeat this process and learn a *policy*—a mapping of observed states to actions—that maximizes the accumulated reward over time [54]. Hence, the terms policy optimization and reinforcement learning are often used interchangeably.

In each timestep, the agent observes the current state $\mathbf{s_t} \in \mathcal{S}$ and executes an action $\mathbf{a_t} \in \mathcal{A}$ using a stochastic policy $\mathbf{a_t} \sim \pi_\theta(\mathbf{a_t} \mid \mathbf{s_t})$, where θ denotes policy parameters. In our case, the policy is a neural network and θ denotes network weights. After that, the agent observes a scalar reward r_t along with the new state $\mathbf{s_{t+1}}$. The goal is to find the optimal policy $\pi_{\theta^*}(\mathbf{a_t} \mid \mathbf{s_t})$ that maximizes the expected return, defined as follows:

$$\mathbb{E}_{\tau \sim \pi_\theta(\mathbf{a} \mid \mathbf{s})} \left[\sum_{t=0}^{T} \gamma^t r_t \right],$$

where $\tau = (\mathbf{s_0}, \mathbf{a_0}, \mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_{T-1}}, \mathbf{a_{T-1}}, \mathbf{s_T})$ is a trajectory generated by starting from $\mathbf{s_0}$ (drawn from an initial state distribution) and following the policy $\pi_\theta(\mathbf{a} \mid \mathbf{s})$ afterwards. A discount factor $\gamma \in [0, 1]$ is used to ensure finite rewards as $T \to \infty$.

In the case of continuous action spaces, policy gradient methods are a common optimization approach. At each optimization iteration, a number of episodes (simulated movement trajectories) are run up to the time horizon T or until encountering a terminal state. The resulting states, actions, and rewards are then used to compute Monte Carlo gradient estimates, and gradient ascent is used to update the policy to improve the expected return.

In this paper, we evaluate our proposed action spaces using three popular policy gradient methods called *Proximal Policy Optimization (PPO)* [55], *Soft Actor-Critic (SAC)* [56], and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [57]. PPO is an on-policy method—i.e., the episode actions are sampled from the policy being optimized—that uses the so-called clipped surrogate loss function or a KL-divergence penalty to allow large but stable policy updates per iteration. SAC is an off-policy method that tries to maximize the expected return while also maximizing policy entropy. TD3 is another off-policy algorithm that improves upon Deep Deterministic Policy Gradient (DDPG) [58] by using two value predictors, updating the policy less frequently than the value predictors, and adding noise to the target actions. PPO, SAC, and TD3 have shown to produce excellent results in computer animation and robotics [56], [57], [59] and are now widely used in popular machine learning frameworks [60], [61], [62], [63].

Advantage Estimation. A concept central to PPO is advantage estimation, which we modify in the PPO variant we use for LLC training (Section 5.3). The advantage of an action $\mathbf{a}$ in state $\mathbf{s}$ denotes how much the action improves

TABLE 1
Details of the OpenAI Gym Environments Used in the Experiments

Environment	Bones	State variables	Action variables
HalfCheetah-v2	7	17	6
Walker2d-v2	7	17	6
Hopper-v2	4	11	3
Humanoid-v2	13	45	17

the expected return, $A^{\pi}(\mathbf{a},\mathbf{s}) = Q^{\pi}(\mathbf{a},\mathbf{s}) - V^{\pi}(\mathbf{s})$, where $V^{\pi}(\mathbf{s})$ is the value function or expected return from state $\mathbf{s}$ following policy π, and $Q^{\pi}(\mathbf{a},\mathbf{s}) = r(\mathbf{a},\mathbf{s}) + \gamma V^{\pi}(\mathbf{s}')$ is the expected return of taking action $\mathbf{a}$ and then following the policy. The $\mathbf{s}'$ denotes the next state resulting from taking action $\mathbf{a}$ from state $\mathbf{s}$. PPO and other advantage-based policy gradient methods adjust the policy parameters to increase the probability of positive advantage actions, and decrease the probability of negative advantage actions. To compute the advantages, one usually trains a separate neural network to predict $V^{\pi}(\mathbf{s})$. This inevitably causes some bias, which is why PPO uses Generalized Advantage Estimation (GAE) [64], a simple procedure that allows trading bias for variance.

4 Simulation Environment

We base our experiments on four challenging MuJoCo [65] environments (agents) in OpenAI Gym [66]: *HalfCheetah-v2*, *Walked2d-v2*, *Hopper-v2*, and *Humanoid-v2*. The details of these environments are shown in Table 1. In order to show how a trained LLC can work across tasks, we defined the following six tasks for each of the environments:

1) *Default*: The default locomotion task in OpenAI Gym MuJoCo environments, where agent receives rewards proportional to speed.
2) *Slow walk*: A locomotion task where the agent is rewarded for staying close to a standing pose while reaching a target velocity of $1m/s$ along the x axis.
3) *Run*: Similar to *slow walk*, but with a target velocity of $4m/s$.
4) *Back walk*: Similar to *slow walk*, but using a target velocity of $-1m/s$.
5) *Balance*: Similar to *slow walk*, but with the target speed of zero.
6) *Stand up*: The characters begin fallen on the ground, and the goal is to stand up.

For tasks 1-5, we utilize the default OpenAI Gym termination criteria: Agents except HalfCheetah-v2 are considered to fail if they fall down, which causes an episode to terminate without reward. Task 6 episodes only terminate after a time limit of 10 seconds.

Task 1 uses the default OpenAI Gym reward function. For tasks 2-6, we define the reward function as

$$r = -\left\|\mathbf{s}_r - \mathbf{g}_r\right\|^2 - 0.01 \times \frac{\|\mathbf{a}\|^2}{N_{DOF}},$$

where $\mathbf{s}_r, \mathbf{g}_r$ are subsets of current and target states used for the reward computation, and $\mathbf{a} \in \mathbb{R}^{N_{DOF}}$ denotes the low-level action (a vector of joint torques) applied in state $\mathbf{s}$. The full agent state $\mathbf{s}$ observed by RL algorithms comprises root vertical position, root velocity, root angular velocity, and joint angles and angular velocities. The subsets $\mathbf{s}_r, \mathbf{g}_r$ used for the reward computation comprise root velocity and joint angles. The target joint angles correspond to a default standing pose. This form of penalizing the deviation from a default pose is a common technique for preventing unnatural movements in optimization-based motion synthesis (e.g., [19], [53]). We believe that these tasks, despite their similarities, provide a wide range of challenges for movement optimization. Similar tasks have also been used in previous work on meta learning [67], [68].

Although the simulation timesteps δ_t for common MuJoCo environments vary, we use a fixed action frequency of $f_a = 10$ Hz in all experiments, repeating each low-level action for $1/(f_a\delta_t)$ simulation steps.

5 Low-Level Controller

We train the LLCs in two steps. First, we use a novel contact-based random exploration method to discover the feasible states and actions of the simulation environment. Then, we utilize an iterative process of value estimation and LLC training. One of the key ideas behind our exploration scheme is resetting the simulation to diverse initial states. This utilizes the fact that in computer animation, as opposed to robotics, manipulating the states can be done without any extra cost. A good example of using this property, which also inspired our episode initialization, is Reference State Initialization of DeepMimic [4]. However, while DeepMimic uses randomly selected states from a motion capture database, our initialization is designed to cover all feasible movement states, without any need for motion data.

5.1 Contact-Based Exploration

Problem Definition. We define the LLC training data generation as an exploration problem: collect data of states, actions, and resulting next states such that: 1) the data covers the joint space of states and actions as completely as possible, and 2) the data enables learning an LLC that allows the agent to efficiently actuate itself to transition between states. Since we are aiming to build task-agnostic LLCs, we refrain ourselves from using any task-dependent reward signals during exploration.

Design Rationale. Our proposed solution focuses on *discovering diverse contact configurations*. This is crucial because a character without an actuated root—e.g., a biped or a quadruped, as opposed to a robot arm mounted on a pedestal—can only affect its environment and actuate its center of mass through contacts. Although contacts are essential for balance and controlled motion, contact discontinuities are what makes learning, and modeling the dynamics hard [16]. Thus, our working hypothesis is that an exploration method designed to discover diverse contact configurations should allow learning better LLCs.

Exploration Algorithm. Our exploration method is detailed in Algorithm 1. The method uses simple random actions while still achieving diverse exploration through the following two principles:

1) The exploration episodes are short ($K = 5$, corresponding to 0.5 seconds of simulation time) to reduce bias towards dynamics attractors such as falling down.
2) The initial state of each episode is randomized to maximize the diversity of contact configurations while ensuring physical plausibility.

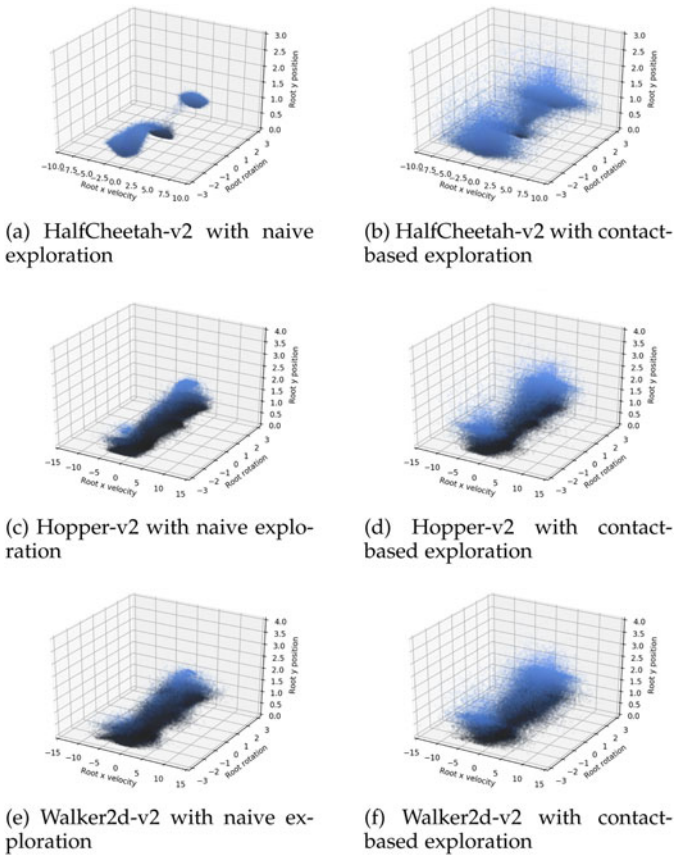
Algorithm 1. Contact-Based Random Exploration

1: **function** EXPLORE(K)
2: **Input**: Rollout horizon K
3: **Output**: Exploration buffer $\mathcal{B}$
4: Initialize exploration buffer $\mathcal{B} \leftarrow \{\}$
5: **while** iteration budget N not exceeded **do**
6: $r \leftarrow$ Random number in $[0, 1]$
7: **if** $r < p_{free}$ **then**
8: $d_{ground} \leftarrow$ Random number in $[0, h_{free}]$
9: **else**
10: **if** $r < p_{free} + p_{close}$ **then**
11: $d_{ground} \leftarrow$ Random number in $[0, h_{close}]$
12: **else**
13: $d_{ground} \leftarrow 0$
14: **end if**
15: **end if**
16: Reset simulation to a random state and locate the character so that it is d_{ground} units above the ground
17: **for** $t = 0, 1, \ldots, K-1$ **do**
18: Observe current state $\mathbf{s_t}$
19: Sample random action $\mathbf{a_t}$
20: $\mathbf{s_{t+1}} \leftarrow$ SIMULATE($\mathbf{s_t}, \mathbf{a_t}$)
21: $\mathcal{B} \leftarrow \mathcal{B} \cup \{[\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}]\}$
22: **end for**
23: **end while**
24: **return** $\mathcal{B}$
25: **end Function**

The initial state randomization proceeds as follows: First, a distance from ground d_{ground} is randomly determined such that the character is either in the air at height h_{free} with probability p_{free}, or close to the ground at height less than h_{close} with probability p_{close}, or on the ground otherwise (Lines 6-15). We use $h_{free} = 1$, $h_{close} = 0.05$, $p_{free} = 0.1$, and $p_{close} = 0.4$ to bias the initialization towards states where the character is in contact with ground or likely to make contact with ground in the next few simulation steps. The rest of the initial state variables are chosen by uniformly selecting the rotation, velocity, and angular velocity, as well as joints angles and angular velocities (Line 16), and adjusting character vertical position so that the distance to ground matches d_{ground}. The valid range for the joint angles and angular velocities is precomputed by keeping the character in the air and actuating the joints with random actions. As humans and many other moving agents spend most of their lives upright and in relatively slow movement, we bias the sampled root rotations and velocities by linearly interpolating towards a default upright pose by a uniformly sampled amount. Examples of the state initialization are shown in the supplementary video, available online.

5.2 State Space Coverage

In order to analyze how our contact-based exploration method in Algorithm 1 covers the state space, we compared it against a naive random exploration baseline, where episodes are initialized using the default OpenAI Gym state initialization (an upright pose with zero intial velocity plus some random noise), and episodes are longer ($K = 100$) to make it possible for the initially non-moving agent to gain velocity and reach diverse states. Such a random exploration approach is used in many recent papers [46], [69], [70], [71].



(a) HalfCheetah-v2 with naive exploration
(b) HalfCheetah-v2 with contact-based exploration
(c) Hopper-v2 with naive exploration
(d) Hopper-v2 with contact-based exploration
(e) Walker2d-v2 with naive exploration
(f) Walker2d-v2 with contact-based exploration

Fig. 2. Scatter plots of visited states when using naive exploration (left) and the proposed contact-based exploration (right). Standing and fallen states are shown in light and dark, respectively. Almost immediately after an episode starts, naive exploration causes the agents to fall down. Contact-based exploration however, uses short-length episodes with randomized state initialization to visit a more diverse set of states.

Fig. 2 shows scatter plots of the visited states using both methods in *HalfCheetah-v2*, *Walked2d-v2*, and *Hopper-v2* environments. The plots visualize the x velocity, rotation, and y position of the root in 100000 visited states. Upright and fallen states are shown in light and dark, respectively. Fig. 2 indicates that the naive random exploration achieves poor state space coverage, with the agent to wasting simulation budget in fallen states. In contrast, our contact-based exploration enables the agent to visit more diverse states.

5.3 Training the Low-Level Controller

We use the exploration data to train a state-reaching LLC, denoted by the policy $\pi_H(\mathbf{a}|\mathbf{s}, \mathbf{G})$ that allows sampling and action $\mathbf{a} \sim \pi_H(\mathbf{a}|\mathbf{s}, \mathbf{G})$ for driving the agent to follow a trajectory of desired next states $\mathbf{G}$ of duration H from the current state $\mathbf{s}$. Later in Section 7, we also study the case where $\mathbf{G}$ only specifies a single target state that is H steps into the future. However, our results show that the former is superior; thus throughout the text $\mathbf{G}$ is assumed to be a state trajectory,

unless otherwise specified. Also note that in case of $H = 1$, the LLC can be considered as an inverse dynamics controller. During LLC training, we use a task-agnostic reward that computes the negated squared deviation from the desired state(s).

We implement the LLCs using multi-layer perceptron (MLP) neural networks that take $\mathbf{s}, \mathbf{G}$ as input, and output a mean and a diagonal covariance matrix for sampling the actions $\mathbf{a}$ from a Gaussian distribution conditioned on $\mathbf{s}, \mathbf{G}$. We train a separate network for each $H = \{1, 2, \ldots, H_{max}\}$, which we found to be more robust than using H as an additional network input. We use $H_{max} = 5$, corresponding to maximum target trajectory length of 0.5 seconds.

Overview. Our LLC training approach is a variant of the Proximal Policy Optimization (PPO) [55] designed according to the following principles:

- To prevent out-of-distribution problems and make LLC robust to anything an HLC might output (e.g., in initial HLC training when the target states output by the HLC may be random and infeasible), we initialize the LLC training episodes using our diverse exploration data, and use a mixture of both feasible and infeasible target state sequences.
- We train the LLC and run our experiments using multiple values for H, assuming that there exists a tradeoff between LLC simplicity and robustness: Low H should allow easier learning and smaller LLC networks, but also reduces the set of states that the LLC can reach. We investigate the effect of H in our experiments in Section 7.
- In training for multiple H values, we assume that a target trajectory of H states can be followed by taking a single action using π_H, and then continuing with π_{H-1}. As detailed below, this allows us to simplify PPO and estimate advantages without a value predictor network.

The LLC training process is detailed in Algorithm 2. The main part is the TrainLLCs() function, which successively trains LLCs for $H = 1, 2, \ldots, H_{max}$. We train each new LLC in $M = 500$ iterations, with the simulation budget of $N = 15000$ actions per iteration ($7.5M$ actions in total).

Supervised Pretraining. For each trained LLC π_H, we first pretrain in a supervised manner so that action mean and variance approximate the exploration data (Line 7). The pretraining uses all subsequences of $H + 1$ states of each exploration episode, using the first subsequence state as the current state, first action as the "ground truth" correct action, and the rest of the states as the target sequence.

Episodic Training. After the pretraining, we continue with PPO, running on-policy episodes of H actions. For each episode, we sample an initial state from the exploration data (Line 10), and target sequence of H states (Line 11). The target state sequence sampling is designed to mostly provide feasible targets from the exploration data, but also include completely random targets to make the LLC robust for anything the HLC outputs and thus prevent out-of-distribution problems. More specifically, we use a mixture of three distributions:

- With probability p_e, we use the state sequence following the initial state in the exploration data. We use $p_e = 0.8$.
- With probability p_l, we uniformly sample a final target state within the state variable ranges in the exploration data, and linearly interpolate the target sequence between the initial and the final state. We use $p_l = 0.1$.
- Otherwise, we use a constant target state sampled uniformly between the minimum and maximum of each state variable in the exploration data.

Appendix E, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2021.3100095, provides an ablation study that compares the above to using only feasible target state sequences, i.e., $p_e = 1.0, p_l = 0$.

Algorithm 2. Training Low-Level Controllers (LLC)

1: **function** TrainLLCs(H_{max})
2: **Input**: LLC horizon H_{max}
3: **Output**: The LLC policies $\pi_{1:H_{max}}$
4: $\mathcal{B} \leftarrow$ Explore(H_{max}) ▷ Algorithm 1
5: Initialize $\pi_{1:H_{max}}$
6: **for** $H = 1, 2, \ldots, H_{max}$ **do**
7: Pretrain $\pi_H(\mathbf{a}|\mathbf{s}, \mathbf{G})$ using exploration buffer $\mathcal{B}$
8: **for** iteration$= 1, 2, \ldots, M$ **do**
9: **while** iteration budget N not exceeded **do**
10: Sample state $\mathbf{s}$ from $\mathcal{B}$
11: Sample target state trajectory $\mathbf{G}_{1:H}$
12: **for** value sample $i = 1, 2, \ldots, N_{adv}$ **do**
13: $\mathbf{a_i}, Q_i \leftarrow CalcQ(\mathbf{s}, H, \mathbf{G}_{1:H}, \pi_{1:H_{max}})$
14: **end for**
15: Estimate state value $V \leftarrow Mean(\{Q_i\})$
16: **for** value sample $i = 1, 2, \ldots, N_{adv}$ **do**
17: Estimate the advantages $A_i \leftarrow Q_i - V$
18: **end for**
19: **end while**
20: Update π_H using advantages $\{A_i\}$ and PPO
21: **end for**
22: **end for**
23: **return** $\pi_{1:H_{max}}$
24: **end Function**
25:
26: **function** CalcQ($\mathbf{s}, H, \mathbf{G}_{1:H}, \pi_{1:H_{max}}$)
27: **Input**: Current state $\mathbf{s}$, LLC horizon H, target state trajectory $\mathbf{G}_{1:H}$, LLC policies $\pi_{1:H_{max}}$.
28: **Output**: First executed action $\mathbf{a}$ when using $\pi_{1:H_{max}}$ to reach $\mathbf{G}_{1:H}$ from $\mathbf{s}$, action value $Q(\mathbf{s}, \mathbf{a})$
29: Sample action $\mathbf{a} \sim \pi_H(\mathbf{a}|\mathbf{s}, \mathbf{G}_{1:H})$
30: $\mathbf{s}' \leftarrow$ Simulate($\mathbf{s}, \mathbf{a}$)
31: Compute the reward $r \leftarrow -\|\mathbf{s}' - \mathbf{G}_1\|^2$
32: **if** $H > 1$ **then**
33: $a', Q' \leftarrow CalcQ(s', H - 1, \mathbf{G}_{2:H}, \pi_{1:H_{max}})$
34: $Q \leftarrow r + Q'$
35: **else**
36: $Q \leftarrow r$
37: **end if**
38: **return** $\mathbf{a}, Q$
39: **end Function**

Modified Advantage Estimation. As the target sequence length decreases over the episode steps, only the first episode action is actually sampled from the π_H being trained, and the rest are determined by the previously trained shorter-horizon LLCs. As a consequence, only the first

action of the episode is used for updating π_H, and the rest of the episode only affects the advantage estimate of the first action. For the first action, we can also directly compute the advantages as $A(\mathbf{a}, \mathbf{s}) = Q(\mathbf{a}, \mathbf{s}) - V(\mathbf{s})$: We run $N_{adv} = 4$ episodes from each initial state (lines 12-14), using the episode return as a single-sample Monte Carlo estimate of Q, returned by the CalcQ() function, which recursively samples the actions and collects rewards using the previously trained LLCs until episode end (Line 13). The rewards are calculated as the negated squared deviation from the target state (Line 31). The mean of the episode returns is used as the Monte Carlo estimate of $V(\mathbf{s})$ (Line 15). Thus, we do not need to train a value predictor network with the episode returns, simplifying the PPO algorithm.

When using a value predictor network, we experienced PPO diverging. A plausible explanation for this is that the highly diverse and sometimes infeasible target states can cause a high variance in the episode returns, which may make the value predictor network unreliable. We further regularize the training by only using actions with positive advantages [72], [73].

6 Movement Optimization Using Low-Level Controllers

The training procedure of the previous section produces a set of H_{max} LLCs that can be used in movement optimization. This means the the action space becomes the space of target state trajectories of length $H \leq H_{max}$. This action space is generic and can be used in offline/online trajectory optimization as well as reinforcement learning, as detailed below.

Offline Trajectory Optimization. In offline trajectory optimization settings, we optimize a trajectory of 4 seconds using CMA-ES [15], i.e., every CMA-ES sample defines a trajectory of $T = 40$ target states. In order to simulate each trajectory, at each timestep we use the next H target states in the trajectory to query the LLC, which computes the low-level action. Similar to the other configurations, each low-level action is repeated for 0.1 seconds. For the last $H - 1$ target states, the target state sequences will be shorter than H steps, and LLCs with smaller H have to be used.

Online Trajectory Optimization. We use a simplified version of Fixed-Depth Informed MCTS (FDI-MCTS) [53], that performs a number of simulation rollouts from the current state using randomly sampled action trajectories of 2 seconds (i.e., trajectory length $T = 20$). The rollout actions are drawn randomly from a Gaussian distribution, with mean equal to the actions of the best trajectory of the previous timestep. For the last rollout action, the previous best trajectory is not available, and the action is sampled uniformly within the action space bounds.

The exploration noise of the rollouts is proportional to the rollout index: $\boldsymbol{\sigma}^2 = \frac{i+1}{N}(M - m)\mathbf{I}$, where i is the current rollout index, N is the total number of rollouts, m and M are the actions' minimum and maximum values, and $\mathbf{I}$ is the identity matrix. The rationale for this is that an online optimizer should both try to refine the current best solution and keep exploring with a large variance, in order to be able to adapt to sudden changes.

The rollouts are simulated forward from the current state in parallel, using multiple threads. After each timestep, badly performing rollouts are terminated, and the simulation resources are reassigned by forking a randomly chosen non-terminated rollout. In effect, the rollouts perform simple tree search. After simulating the rollouts up to the planning horizon, the first action of the best rollout is returned as the approximately optimal action.

Reinforcement Learning. For reinforcement learning, we use Proximal Policy Optimization (PPO) [55], Soft Actor-Crit (SAC) [56], and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [57], three popular RL algorithms for continuous control.

Implementation Details. For the LLCs we used multi-layer perceptron (MLP) neural networks using Tensorflow [74] and the Swish activation function [75]. For *HalfCheetah-v2*, *Walked2d-v2*, and *Hopper-v2*, the LLC policies had 2 hidden layers of 64 neurons, and for *Humanoid-v2* we used larger policies with 3 hidden layers of 128 neurons. While building the exploration buffer and training the low-level controllers, we bypassed the default termination conditions defined in OpenAI Gym to make sure that the exploration is diverse and task-agnostic. The RL training uses PPO, SAC, and TD3 implementations from the *stable-baselines* repository [76]. The hyperparameters used in the experiments are given in Appendix A, available in the online supplemental material.

7 Experiments

This section empirically compares the different action spaces discussed above. To reiterate, we compare the following:

- *Baseline*: The default action space without using low-level controllers, i.e., joint torques.
- *LLC[Naive]*: The action space posed by the LLCs trained using the naive exploration approach with less diverse episode initialization.
- *LLC[Contact-Based]*: The action space posed by the LLCs trained using our proposed contact-based exploration approach.

The LLCs are queried with a sequence of target states, H states in total. For completeness, we also tested a version where a single target state was given to be reached H steps in the future. However, as detailed in Appendix B, available in the online supplemental material, this results in overall worse performance.

The experiments below are designed to answer the following research questions:

- *RQ1*: Which action space works best, on average? Does using the LLC and training with the proposed contact-based exploration method improve optimization efficiency?
- *RQ2*: How does the LLC planning horizon H affect the optimization? Which H should one use?
- *RQ3*: How consistent are the results across the various tasks, agents, and optimization approaches?
- *RQ4*: How does using the LLC affect the optimization landscape? Why does it make optimization easier?

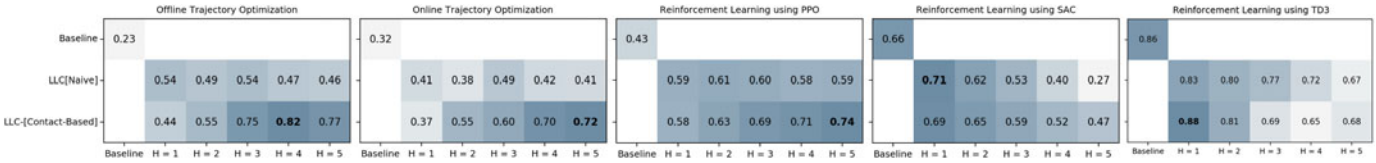


Fig. 3. Comparing different action spaces (baseline versus LLC versions) and H values in terms of normalized average episode/trajectory returns (higher is better) in the four optimization approaches. Overall, using an LLC outperforms the baseline, and LLCs trained using our contact-based exploration outperform LLCs trained using naive random exploration.

To ensure that the results are generalizable, the experiments are performed across the four agents and six tasks described in Section 4 as well as the four different optimization approaches described in Section 6.

7.1 Comparing Action Spaces

Fig. 3 shows the benchmark results using normalized aggregate scores (higher is better) that measure the overall performance of each action space and optimization method across all agents and tasks. Using the LLC generally leads to significantly better results than the baseline, except with SAC and TD3, where the improvement is only marginal. Furthermore, LLCs trained with contact-based exploration in Algorithm 1 mostly outperform the ones trained using naive exploration.

To obtain the results, we performed 10 independent optimization runs with different random seeds for each combination of agent, task, optimizer, action space, and LLC horizon H. In total, this yields almost 20000 optimization runs. The simulation budget used in reinforcement learning using PPO, SAC, and TD3 is $10M$ simulation timesteps for *Humanoid-v2* and $1M$ simulation timesteps for the other three environments. After all runs, the mean episode returns of the final iterations were normalized over each agent-task pair so that the worst run had score 0 and the best run had score 1. The scores in Fig. 3 are averages of the normalized returns over all tasks and agents, i.e., a total of $10 \times 4 \times 6 = 240$ runs per cell, which should yield reasonably reliable results.

7.2 Effect of H

RQ2 concerns the LLC horizon H, which is the most important tuning parameter in our approach and also determines the number of LLCs to be trained in Algorithm 2.

As it can be seen in Fig. 3, three out of five approaches (offline/online trajectory optimization and PPO) work better with higher H values, i.e., $H = 4$ and $H = 5$. In contrast, SAC and TD3 score highest with $H = 1$ and their performance declines consistently when H grows. We did not perform an exhaustive evaluation using $H > 5$, as in our early experiments, further increasing H did not improve the results with any of the optimization approaches.

7.3 Consistency Across Tasks and Agents

The aggregate scores in Fig. 3 do not allow comparing the consistency and variability of the results. Thus, to answer RQ3, we also scrutinized the performance of each agent and task. We simplified the investigation by only considering the best-performing H, i.e., $H = 4$ for offline trajectory optimization, $H = 5$ for online trajectory optimization and PPO, and $H = 1$ for SAC and TD3.

We outline the findings below, and Fig. 4 shows example convergence plots from the Hopper-v2 environment. Full results of all environments can be found in Appendix C, available in the online supplemental material.

Offline Trajectory Optimization. Both LLC versions are clearly better than the baseline in all tasks except for the stand up task. However, even in this case the final performance is comparable to the baseline. Overall, the contact-based exploration clearly surpasses the naive version.

Online Trajectory Optimization. The results are consistent with those obtained in offline trajectory optimization: Both LLC-based approaches improve over the baseline, except in the stand up task. Again, the version with the contact-based exploration beats the one with naive exloration.

Reinforcement Learning Using PPO. Again, LLC with contact exploration performs best in all tasks, including the stand up task, but the difference is less pronounced with the humanoid agent.

Reinforcement Learning Using SAC. In this configuration, all action spaces perform roughly similarly, except for a few tasks. LLC yields significant gains in the hopper default and humanoid stand-up tasks, but performs worse in the humanoid default task.

Reinforcement Learning Using TD3. Similar to SAC, there are no major differences between the action spaces.

The overall worse results of the LLC with the humanoid agent—while still providing significant gains with PPO and offline trajectory optimization—are likely due to both the higher state dimensionality and the aggressive episode termination whenever the agent starts to fall down. The high dimensionality makes the exploration data more sparse, and because the termination limits the agent to a small subregion of the state space, the contact-based diverse exploration can be expected to yield less gains. Fittingly, the LLC performs best with SAC in the stand up task which does not use termination.

To summarize the findings in Fig. 4, our approach is particularly effective in both offline and online trajectory optimization settings as well as reinforcement learning using PPO.

7.4 Effect of LLC on the Optimization Landscape

So far, our results show a clear advantage of using the LLC, but the data is insufficient for analyzing the underlying reasons. A plausible explanation for trajectory optimization is that when not using the LLC and using control torques as actions, a small change to a single action can cause a large divergence in the rest of the trajectory, e.g., making a bipedal character fall. Furthermore, as pointed out in the case of a simple inverted pendulum in [11], the sum of rewards or costs over the trajectory is more sensitive to the

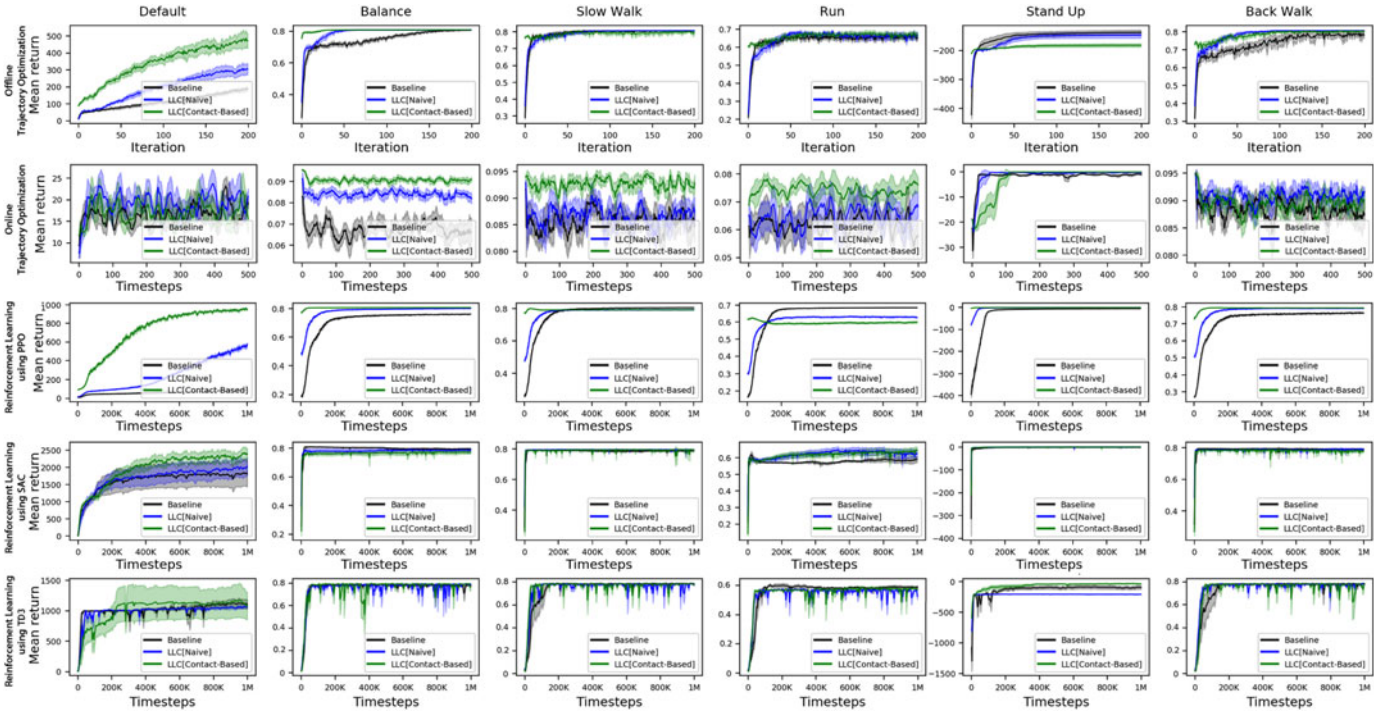


Fig. 4. Results of offline/online trajectory optimization and reinforcement learning (using PPO , SAC, and TD3) in the Hopper-v2 environment. Each row corresponds to a different movement optimization approach, and each column corresponds to a different movement task.

first actions, which makes the optimization ill-conditioned, exhibiting elongated optima where an optimizer slowly zigzags between the cost function walls, or along a reward function ridge. In contrast, when using the LLC, perturbing the action of a single timestep—i.e., a single target state in a sequence—has only a small effect on states and rewards of the other timesteps.

In light of the above, optimizing without LLC should exhibit much narrower and/or elongated optima which slow down the convergence. This is a hypothesis we can test using the random 2D landscape slice visualization approach of [77], also applied to movement optimization by [11]. This means that a multidimensional objective function is characterized by plotting it around the found optimum along a 2D slice of the parameter space defined by a pair of random orthogonal 2D subspace basis vectors. Fig. 5 shows such 2D slices of offline trajectory optimization of the hopper balance and walker run tasks. As can be seen in the figure, using low-level controllers leads to smoother optimization landscapes with large and less ill-conditioned high-return basins. A similar visualization in the case of reinforcement learning can be found in Appendix D, available in the online supplemental material.

8 Limitations and Future Work

Considering the results, a clear limitation of our approach is that although we are able to improve the efficiency of trajectory optimization as well as on-policy RL using PPO, the improvements are only marginal in off-policy RL using SAC and TD3. When using SAC or TD3 and the best-performing choice for H, the compared action spaces yield approximately similar results in almost all the tasks. Future work is needed to investigate why this is the case. We hypothesize that RL algorithm differences in value estimation and learning play a role. In PPO, this is based on episode returns, which are more sensitive to small action perturbations when not using the LLC, similar to the trajectory returns of trajectory optimization. SAC and TD3's Bellman backup using the twin Q networks might be less affected by the choice of action space.

In trajectory optimization, the stand up task did not benefit from using the LLC. One explanation for this is that the task requires more aggressive movement than the other tasks, using the LLC appears to bias the agent's initial movement exploration towards smooth and continuous movements. On the other hand, the problem does not persist in policy optimization.

A further limitation is that we assume a full target observation is known for the LLC. Learning an HLC or directly using an LLC—e.g., for keyframe animation purposes—could be easier if one could also express the importance of each target observation variable. For example, one could produce locomotion by specifying a desired root rotation and velocity, and leave the other degrees of freedom to be decided by the LLC. We are currently investigating this, e.g., through the incorporation of recent machine learning models that can infer any number of unspecified variables [78]. Another potential topic for future work could be to use our contact-based exploration method in offline RL methods such as conservative Q-learning (CQL) [79]. The added stability provided by an LLC could make it easier to learn from limited data.

Finally, future work is needed to extend the initial state randomization of our contact-based exploration algorithm to object manipulation tasks, and improve the robustness of the LLCs. In the current version, the environment is assumed to be static, which is not valid for, e.g., real-world

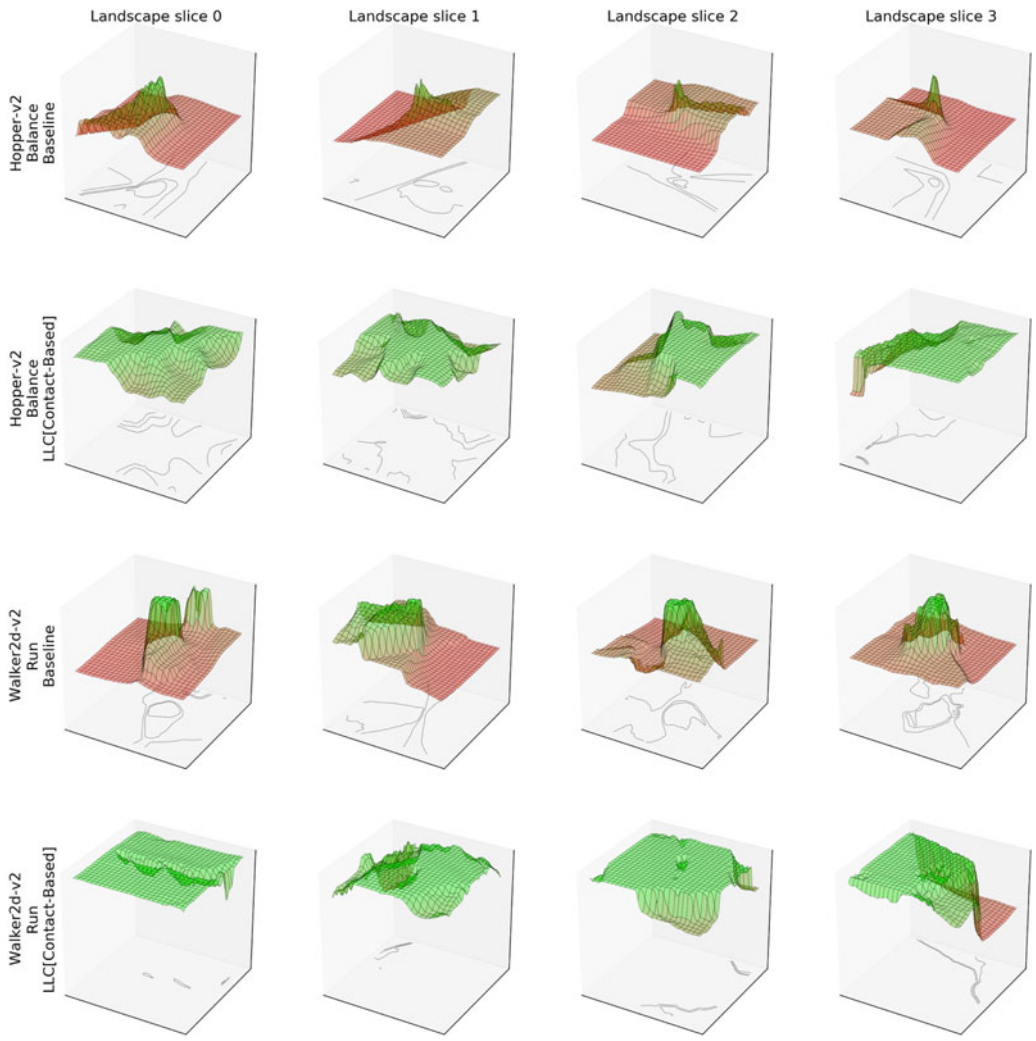


Fig. 5. Offline trajectory optimization landscapes of Walker2d-v2 running and Hopper-v2 balancing with/without the low-level controllers (the up-axis shows the average episode return, i.e., higher is better). Using low-level controllers leads to smoother landscapes with significantly wider high-return basins that are easy to find using sampling-based optimization.

robotic manipulation tasks. Our LLCs are also not robust enough for following any target trajectory without errors, and the HLC has to compensate for the LLC's imperfections.

9 Conclusion

We have proposed a hierarchical movement control approach where a low-level controller (LLC) is trained to follow target state trajectories, and movement optimization—including both trajectory optimization and reinforcement learning—can then operate in the space of target states instead of low-level actions such as control torques. Through extensive experiments across multiple agents, tasks, and optimization methods, we have shown that our LLCs improve movement optimization convergence and the attainable objective function values. This can be explained by the LLC making state-space movement trajectories less sensitive to small changes of the actions, which results in wider optima that are easier to find.

In contrast to previous work that trains LLCs using motion capture data and is thus limited in terms of supported agents and movements, we have proposed a simple contact-based exploration method to synthesize diverse, task-agnostic LLC training data for a variety of agents that depend on contact forces for actuation, e.g., bipeds and quadrupeds. Our experiments also indicate that LLCs trained with our exploration data perform better than LLCs trained with baseline random exploration data. We believe our work provides a building block towards a general solution to the movement optimization problem, improving a wide range of movement optimization applications.

Acknowledgments

This work was supported by Academy of Finland under Grant 299358 and the Technology Industries of Finland Centennial Foundation. The experiments utilized the Triton cloud computing infrastructure of Aalto University. Part of the research was conducted while the first author was a visiting researcher at the University of British Columbia, Canada.

References

[1] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[2] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Discovering and synthesizing humanoid climbing movements," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 43:1–43:11, Jul. 2017. [Online]. Available: http://doi.acm.org/10.1145/3072959.3073707

[3] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, "DReCon: Data-driven responsive control of physics-based characters," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–11, 2019.

[4] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 143:1–143:14, Jul. 2018. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201311

[5] J. Won, D. Gopinath, and J. Hodgins, "A scalable approach to control diverse behaviors for physically simulated characters," *ACM Trans. Graph.*, vol. 39, no. 4, 2020, Art. no. 33.

[6] X. B. Peng and M. van de Panne, "Learning locomotion skills using DeepRL: Does the choice of action space matter?," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, 2017, pp. 1–13.

[7] J. Merel, D. Aldarondo, J. Marshall, Y. Tassa, G. Wayne, and B. Olveczky, "Deep neuroethology of a virtual rodent," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SyxrxR4KPS

[8] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, 2017, Art. no. 41.

[9] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," in *Proc. Int. Conf. Learn. Representations*, 2019.

[10] S. Lee, M. Park, K. Lee, and J. Lee, "Scalable muscle-actuated human simulation and control," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–13, 2019.

[11] P. Hämäläinen, J. Toikka, A. Babadi, and K. Liu, "Visualizing movement control optimization landscapes," *IEEE Trans. Vis. Comput. Graphics*, to be published, doi: 10.1109/TVCG.2020.3018187.

[12] K. Yin, K. Loken, and M. Van de Panne, "SIMBICON: Simple biped locomotion control," *ACM Trans. Graph.*, vol. 26, no. 3, 2007, Art. no. 105–es.

[13] K. Yin, S. Coros, P. Beaudoin, and M. van de Panne, "Continuation methods for adapting simulated skills," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–7, 2008.

[14] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, no. 6, 2013, Art. no. 206.

[15] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*. Berlin, Germany: Springer, 2006, pp. 75–102.

[16] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, 2012, Art. no. 43.

[17] K. Naderi, A. Babadi, and P. Hämäläinen, "Learning physically based humanoid climbing movements," *Comput. Graph. Forum*, vol. 37, no. 8, pp. 69–80, 2018.

[18] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen, "Online motion synthesis using sequential Monte Carlo," *ACM Trans. Graph.*, vol. 33, no. 4, 2014, Art. no. 51.

[19] A. Babadi, K. Naderi, and P. Hämäläinen, "Intelligent middle-level game control," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[20] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1993, pp. 271–278.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1/2, pp. 181–211, 1999.

[22] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.

[23] X. B. Peng, G. Berseth, and M. van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 80:1–80:11, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2766910

[24] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, 2016, Art. no. 81.

[25] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep Q-learning," *ACM Trans. Graph.*, vol. 36, no. 3, pp. 1–14, 2017.

[26] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015, Art. no. 529.

[27] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: Learning composable hierarchical control with multiplicative compositional policies," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 3681–3692.

[28] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," 2018, *arXiv:1810.01257*.

[29] M. Andrychowicz *et al.*, "Hindsight experience replay," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5048–5058.

[30] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 3303–3313.

[31] J. Merel *et al.*, "Learning human behaviors from motion capture by adversarial imitation," 2017, *arXiv:1707.02201*.

[32] Y.-S. Luo, J. H. Soeseno, T. P.-C. Chen, and W.-C. Chen, "CARL: Controllable agent with reinforcement learning for quadruped locomotion," *ACM Trans. Graph.*, vol. 39, no. 4, 2020, Art. no. 38.

[33] J. Merel *et al.*, "Hierarchical visuomotor control of humanoids," 2018, *arXiv:1811.09656*.

[34] J. Merel *et al.*, "Catch & carry: Reusable neural controllers for vision-guided whole-body tasks," *ACM Trans. Graph.*, vol. 39, no. 4, 2020, Art. no. 39.

[35] J. Merel *et al.*, "Neural probabilistic motor primitives for humanoid control," 2018, *arXiv:1811.11711*.

[36] L. Hasenclever, F. Pardo, R. Hadsell, N. Heess, and J. Merel, "CoMic: Complementary task learning & mimicry for reusable skills," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4105–4115.

[37] S. Park, H. Ryu, S. Lee, S. Lee, and J. Lee, "Learning predict-and-simulate policies from unorganized human motion data," *ACM Trans. Graph.*, vol. 38, no. 6, 2019, Art. no. 205.

[38] S. Levine and V. Koltun, "Guided policy search," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1–9.

[39] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," *Robot., Sci. Syst.*, vol. 4, 2014.

[40] J. Rajamäki and P. Hämäläinen, "Augmenting sampling based controllers with machine learning," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, 2017, Art. no. 11.

[41] J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: Example-guided control of flapping flight," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 1–13, 2017.

[42] A. Babadi, K. Naderi, and P. Hämäläinen, "Self-imitation learning of locomotion movements through termination curriculum," in *Proc. Motion Interact. Games*, 2019, pp. 1–7.

[43] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[44] Ł. Kaiser *et al.*, "Model based reinforcement learning for Atari," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1xCPJHtDB

[45] C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 504–511.

[46] R. Boney, J. Kannala, and A. Ilin, "Regularizing model-based planning with energy-based models," in *Proc. Conf. Robot Learn.*, 2019.

[47] L. Orseau, T. Lattimore, and M. Hutter, "Universal knowledge-seeking agents for stochastic environments," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2013, pp. 158–172.

[48] T. Hester and P. Stone, "Intrinsically motivated model learning for developing curious robots," *Artif. Intell.*, vol. 247, pp. 170–186, 2017.

[49] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1471–1479.

[50] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," 2019, *arXiv:1912.01603*.

[51] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," 2020, *arXiv:2005.05960*.

[52] L. Liu, K. Yin, M. van de Panne, and B. Guo, "Terrain runner: Control, parameterization, composition, and planning for highly dynamic motions," *ACM Trans. Graph.*, vol. 31, no. 6, 2012, Art. no. 154.
[53] J. Rajamäki and P. Hämäläinen, "Continuous control Monte Carlo tree search informed by multiple experts," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 8, pp. 2540–2553, Aug. 2019.
[54] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[56] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*.
[57] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
[58] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[59] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," 2020, *arXiv:2004.00784*.
[60] P. Dhariwal *et al.*, "OpenAI baselines," 2017. [Online]. Available: https://github.com/openai/baselines
[61] D. Hafner, J. Davidson, and V. Vanhoucke, "Tensorflow agents: Efficient batched reinforcement learning in tensorflow," 2017, *arXiv:1709.02878*.
[62] A. Juliani *et al.*, "Unity: A general platform for intelligent agents," 2018, *arXiv:1809.02627*.
[63] F. Pardo, "Tonic: A deep reinforcement learning library for fast prototyping and benchmarking," 2020, *arXiv:2011.07537*.
[64] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.
[65] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.
[66] G. Brockman *et al.*, "OpenAI gym," 2016, *arXiv:1606.01540*.
[67] N. Heess *et al.*, "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.
[68] A. Gupta, B. Eysenbach, C. Finn, and S. Levine, "Unsupervised meta-learning for reinforcement learning," 2018, *arXiv:1806.04640*.
[69] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4754–4765.
[70] D. Ghosh *et al.*, "Learning to reach goals via iterated supervised learning," 2019.
[71] K. Schmeckpeper *et al.*, "Learning predictive models from observation and interaction," 2019.
[72] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *Proc. IEEE Int. Symp. Approx. Dyn. Program. Reinforcement Learn.*, 2007, pp. 272–279.
[73] P. Hämäläinen, A. Babadi, X. Ma, and J. Lehtinen, "PPO-CMA: Proximal policy optimization with covariance matrix adaptation," 2018, *arXiv:1810.02541*.
[74] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," *Software available from tensorflow.org*, 2015. [Online]. Available: https://www.tensorflow.org/
[75] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.
[76] A. Hill *et al.*, "Stable baselines," 2018. [Online]. Available: https://github.com/hill-a/stable-baselines
[77] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6389–6399.
[78] P. Hämäläinen and A. Solin, "Deep residual mixture models," 2020, *arXiv:2006.12063*.
[79] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," 2020.

Amin Babadi is currently working toward the doctoral degree with the Department of Computer Science, Aalto University, Espoo, Finland. His research focuses on developing efficient, creative movement artificial intelligence for physically simulated characters in multi-agent settings. He has previously worked on three commercial games, developing AI, animation, gameplay, and physics simulation systems.

Michiel van de Panne received the BASc degree from the University of Calgary, Calgary, Canada, in 1987, and the MASc and PhD degrees from the University of Toronto, Toronto, Canada, in 1989 and 1994, respectively. From 1993 to 2001, he was a faculty member with the Department of Computer Science, University of Toronto. Since 2002, he has been with the Department of Computer Science, University of British Columbia as associate professor (2002–2008) and as full professor (2008). He served as associate head for Research and Faculty Affairs during 2011–2014. During 2000-2001, he was a visiting professor with the University of British Columbia, and founded Motion Playground Inc. to develop games and educational applications using physics-based animation and simulation. He was a visiting researcher with INRIA Sophia Antipolis during 2007–2008.

C. Karen Liu received the PhD degree in computer science from the University of Washington, Seattle, Washington. She is an associate professor with the Department of Computer Science, Stanford University. Her research interests include computer graphics and robotics, including physics-based animation, character animation, optimal control, reinforcement learning, and computational biomechanics. She developed computational approaches to modeling realistic and natural human movements, learning complex control policies for humanoids and assistive robots, and advancing fundamental numerical simulation and optimal control algorithms. The algorithms and software developed in her lab have fostered interdisciplinary collaboration with researchers in robotics, computer graphics, mechanical engineering, biomechanics, neuroscience, and biology. She received a National Science Foundation CAREER Award, an Alfred P. Sloan Fellowship, and was named Young Innovators Under 35 by Technology Review. In 2012, she received the ACM SIGGRAPH Significant New Researcher Award for her contribution in the field of computer graphics.

Perttu Hämäläinen received the MSc(Tech) degree from the Helsinki University of Technology, Espoo, Finland, in 2001, the MA degree from the University of Art and Design Helsinki, Helsinki, Finland, in 2002, and the doctoral degree in computer science from the Helsinki University of Technology, Espoo, Finland, in 2007. Currently, he is an associate professor with Aalto University, publishing on human-computer interaction, computer animation, machine learning, and game research. He is passionate about human movement in its many forms, ranging from analysis and simulation to first-hand practice of movement arts such as parkour or contemporary dance.