Self-Imitation Learning of Locomotion Movements through Termination Curriculum

Amin Babadi, Kourosh Naderi, and Perttu Hämäläinen
{amin.babadi,kourosh.naderi,perttu.hamalainen}@aalto.fi
Department of Computer Science, Aalto University
Helsinki, Finland

Figure 1: Example cycles synthesized in the reference motion generation stage for Mech (22 frames), Orc (19 frames), and Wolf (32 frames). Due to lack of space, five frames of each cycle are shown.

ABSTRACT
Animation and machine learning research have shown great advancements in the past decade, leading to robust and powerful methods for learning complex physically-based animations. However, learning can take hours or days, especially if no reference movement data is available. In this paper, we propose and evaluate a novel combination of techniques for accelerating the learning of stable locomotion movements through self-imitation learning of synthetic animations. First, we produce synthetic and cyclic reference movement using a recent online tree search approach that can discover stable walking gaits in a few minutes. This allows us to use reinforcement learning with Reference State Initialization (RSI) to find a neural network controller for imitating the synthesized reference motion. We further accelerate the learning using a novel curriculum learning approach called Termination Curriculum (TC), that adapts the episode termination threshold over time. The combination of the RSI and TC ensures that simulation budget is not wasted in regions of the state space not visited by the final policy. As a result, our agents can learn locomotion skills in just a few hours on a modest 4-core computer. We demonstrate this by producing locomotion movements for a variety of characters.

CCS CONCEPTS
• Computing methodologies → Reinforcement learning; Procedural animation; Robotic planning; Physical simulation; Continuous space search; Neural networks.

KEYWORDS
Continuous Control, Physically-Based Animation, Online Optimization, Reinforcement Learning, Policy Optimization, Self-Supervised Learning

ACM Reference Format:
Amin Babadi, Kourosh Naderi, and Perttu Hämäläinen. 2019. Self-Imitation Learning of Locomotion Movements through Termination Curriculum. In Under review for ACM/SIGGRAPH conference on Motion, Interaction and Games. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION
Intelligent control of physics simulation is an increasingly popular approach for synthesizing physically-plausible animations for simulated characters. This requires a method that outputs, for each timestep, simulation actuation parameters such as joint torques...
such that the character performs some desired movement. This poses a continuous control problem with high state and control dimensionality, where the environment is governed by complex physical interactions. Physically-based animation has applications in games, simulation, and robotics [7].

Current approaches for solving physically-based animation can be divided into two categories: 1) planning and search, and 2) reinforcement learning. Planning and search methods use the interleaved mechanism of iterative generation and evaluation of candidate solutions until finding a sufficiently good one [15]. On the other hand, reinforcement learning (RL) methods learn how to act through interaction with the environment [5].

Although RL methods have shown great potential in learning complex skills [34], they often fail in producing smooth and believable motions1. This has recently been solved by a framework called DeepMimic, by showing that in the presence of high-quality pre-recorded animations, RL methods are able to learn a wide range of skills with near-optimal motion quality [23]. The immediate extension to this approach, called SFV, is designed to work with videos, as an alternative resource for automated extraction of reference animations [27].

Despite the impressive results produced by DeepMimic and SFV, their dependency on high-quality movement recordings limits the types of characters or behaviors they can support. For example, providing motion capture animations or video recordings for characters with ad hoc rigs is almost impossible, and producing hand-designed animations for such character is expensive and time-consuming. Plus, using real-life videos for producing reference motions is not very suitable for games, where exaggeration in movements is frequently used to encourage the feeling of empowerment for the players [9]. Last but not least, such data-driven approaches can only be used for synthesizing movements whose reference motion is available; so they cannot be used for producing novel movements. This calls for more general approaches that can work with ad hoc characters and movements.

In this paper, we propose a self-imitation learning approach for enabling rapid learning of stable locomotion controllers. Essentially, our approach combines FDI-MCTS [29] and DeepMimic [23], two recent methods for continuous control. FDI-MCTS is an online tree search method that is able to produce high-quality locomotion movements in just a few minutes [29]. We are motivated to mitigate the main limitations of both methods, namely the high run-time cost of FDI-MCTS and the data-dependency of DeepMimic.

We begin by using FDI-MCTS to generate a cyclic locomotion movement as the reference motion (examples of synthesized motions are shown in Fig. 1). Then we employ a training mechanism similar to DeepMimic, to find a neural network controller that imitates the reference motion. We also propose Termination Curriculum (TC), a novel curriculum learning approach for accelerating the imitation learning. Our experiments show that our approach is able to learn robust locomotion skills for a broad set of 3D characters. All controllers are trained in less than four hours of CPU time, which is significantly faster than DeepMimic and SFV.

The rest of this paper is organized as follows. Section 2 overviews the previous approaches for synthesizing physically-based animations. In Section 3, the basic preliminary concepts of online optimization and policy optimization are introduced. After that, our proposed approach is explained in detail in Section 4. Section 5 covers the setup and results of our experiments. Finally, at Section 6, we overview the limitations and future work to our approach.

2 RELATED WORK
Our work aims at producing locomotion movements for physically-based characters. There has been a large body of research on this problem, especially after remarkable breakthroughs in Deep Reinforcement Learning (DRL) [20, 35, 37]. Proposed approaches can be divided into two categories: planning and search (Section 2.1), and reinforcement learning (Section 2.2). Some approaches also employ reference animations to produce believable movements. We refer to this technique as motion imitation (Section 2.3).

2.1 Planning and Search
Planning and search approaches use the classic-style search for optimizing movements. The main pipeline behind these approaches includes the following three steps: 1) A number of random trajectories are generated, 2) Each trajectory is evaluated by forward simulation and computing a cost function, and 3) The trajectory with minimum cost is picked as the solution. This has resulted in a flexible and powerful mechanism for solving optimization problems for a long time.

Evolutionary Strategies (ES) are a family of black-box optimization methods that are also very easy to use in parallel [30]. One of the most common ES methods is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [13]. CMA-ES has been used in an offline manner to learn the parameters for controller of physically-simulated characters [8]. Another study has used CMA-ES as an offline low-level controller for synthesizing humanoid wall climbing movements [22]. It has been shown that rolling horizon version of CMA-ES can be used in different real-time scenarios [17, 31]. Recent studies have used CMA-ES for synthesizing sports movements in a two-player martial arts interface [3] and single-agent basketball [18].

Monte Carlo methods have received considerable interest in domains where the search budget is limited. Sequential Monte Carlo (SMC) has been shown to be effective for online synthesis of physically-based animations [12]. It has also been used for replicating motion capture data by breaking the problem into a sequence of control fragments [19]. Another Monte Carlo method, called Monte Carlo Tree Search (MCTS) [6], has shown great performance in real-time applications and games [38]. MCTS has been used in Alpha Go Zero for improving the policy using self-play [37]. Fixed Depth Informed MCTS (FDI-MCTS) is a continuous version of MCTS used in a physically-based control. FDI-MCTS uses a policy network trained with supervised learning to reduce the movement noise produced by the sampling-based controller [29].

2.2 Reinforcement Learning
Reinforcement Learning (RL) is a learning process, conducted through interaction between an agent and an environment, where the goal

---

1 See, for example, https://www.youtube.com/watch?v=faDKMm0wOSQ&
is to maximize the reward by optimizing the agent's actions [40]. RL methods have become significantly more powerful in the recent years, mainly after the Deep Reinforcement Learning (DRL) success in Atari games [20] and the game of Go [35, 37]. Recently, this approach was also used in AlphaZero, a system with superhuman performance in the games of go, chess, and shogi [36].

It has been shown that when the motion is guided by a finite state machine, actor-critic methods can be used for learning train-adaptive locomotion skills [24, 25]. Actor-critic methods have also been successfully used in hierarchical controllers [26]. Deep Deterministic Policy Gradient (DDPG) [16] has been recently used for learning arm control policies for basketball dribbling [18].

Two of the most common RL algorithms are called Trust Region Policy Optimization (TRPO) [32] and Proximal Policy Optimization (PPO) [34]. The key element in these methods is a surrogate objective function that allows for more than one gradient update per data sample. PPO has shown to have tendency of prematurely shrinking the exploration variance, which makes it prone to getting stuck in local optima [11]. However, studies have shown that PPO outperforms TRPO in most cases, which makes it the dominant algorithm used in continuous control [23, 41]. In this paper, we use PPO as the base RL algorithm for learning locomotion movements.

Curriculum learning is the process of learning a series of tasks in increasing order of complexity [4]. It is a powerful technique for improving learning performance in terms of both convergence speed and output quality. This technique has been used for learning humanoid climbing movements, where the agent learns 1-, 2-, 3-, and 4-limb movements in order (a limb can be either one of agent’s hands or feet) [21]. A recent study proposed a continuous curriculum learning method for providing physical assistance to help the character in locomotion movements [41]. Our work uses a similar curriculum learning mechanism for imitating a reference animation.

### 2.3 Motion Imitation

One of the main approaches for synthesizing animations is to guide the process using available animation data, which is usually designed manually or recorded using motion capture. A recent study proposed a method for training physically-based character controllers using motion capture animations [19]. In the context of kinematic-based control, another study has introduced a neural network architecture whose weights can be computed using cyclic functions with respect to a dataset of pre-recorded animations [14]. A generalization of this architecture has been successfully used for synthesizing robust quadruped movements [42].

Another popular data-driven approach for solving physically-based control is to learn through imitation. The first method in this category was called DeepLoco, a system that trains high-level and low-level controllers such that the low-level controller is encouraged to imitate a reference animation [26]. A descendant of this method, called DeepMimic, has been able to produce a wide set of high-quality and robust movements by imitating a large motion capture dataset [23]. The most recent variant of this method is able to extract reference animations directly from videos, which makes the training pipeline significantly cheaper [27]. The imitation learning process employed in our work differs from the one used in DeepMimic as in our work the reference animations are automatically synthesized without using any animation or video. That is the reason we use the term self-imitation learning for this process.

### 3 PRELIMINARIES

This section covers the basics of online optimization and policy optimization methods. We follow the same notation used in relevant previous works, especially the notation introduced in [23].

#### 3.1 Online Optimization

Online optimization is one of the main approaches for generating physically-based animations [7]. The idea is to generate a set of candidate solutions (i.e., action sequences), and find the most cost-efficient solution by evaluating them. This is usually done using forward simulation until some time horizon $H$ and computing some cost function $\mathcal{J}$. If the character’s current state is $s_t \in S$, forward simulation of a sequence of actions $a_1, a_2, \ldots, a_t \in \mathcal{A}$ leads to a trajectory $O = (s_1, a_1, s_2, a_2, \ldots, s_H, a_H)$ (the superscript $O$ stands for Online optimization). Then, the problem will be to find the action sequence that minimizes the accumulative cost, i.e.,

$$\mathcal{J}(O) = \sum_{t=1}^{H-1} \left[ C_{\mathcal{A}}(a_t) + C_{\mathcal{S}}(s_{t+1}) \right],$$

where $C_{\mathcal{A}}$ and $C_{\mathcal{S}}$ are functions for computing the state and action costs, respectively. $C_{\mathcal{S}}$ usually encodes some information about the target movement. In the walking task for example, the character should keep its center of mass above its feet and its mean velocity close to some desired walking velocity. $C_{\mathcal{A}}$ usually penalizes the amount of torque applied to each body joint in order to avoid extreme movements [28].

We use a recent open-source online optimization method called Fixed-Depth Informed Monte Carlo Tree Search (FDMI-MCTS) [29] to produce a cyclic locomotion movement as the reference motion. FDMI-MCTS synthesizes movements using an interleaved process of tree search and supervised learning, as illustrated in Fig. 2. The supervised learning component is trained using the best controls found during the tree search in order to reduce the noise in produced movements. This results in emergence of stable locomotion gaits for different types of characters in less than a minute of CPU time, allowing rapid cost function design iteration. Movement is initially noisy, but running the algorithm for a few more minutes removes the noise. Next, we briefly explain how the tree search and supervised learning components work in FDMI-MCTS. More details about this method along with its implementation details can be found in [29].

FDMI-MCTS uses a variant of Monte Carlo tree search (MCTS) by sampling $N$ random control trajectories (i.e., a series of target angles) for $H$ timesteps. At each timestep, it prunes the trajectories whose costs are more than some adaptive threshold, and replaces them with duplicates of the trajectories low cost. After forward simulation of all $N$ trajectories for $H$ timesteps, the best found

---

1https://github.com/JooseRajamaeki/TVCG18
The basic definition of reinforcement learning includes an agent that interacts with some environment, and its goal is to maximize the accumulated rewards over time. Policy optimization refers to a family of reinforcement learning methods, in which the goal is to optimize the agent’s policy with respect to the expected return. The policy \( \pi_\theta (a \mid s) \) is usually modeled using a neural network, parameterized by \( \theta \), and defines a mapping from a state \( s \in S \) to a distribution over actions \( a \in A \).

In each timestep, the agent observes the current state \( s_t \) and samples its next action \( a_t \) from the distribution \( \pi_\theta (a_t \mid s_t) \). After that, the environment is updated and the agent observes a scalar reward \( r_t \) along with the new state \( s_{t+1} \). The goal is to find the optimal parameters \( \theta^* \) that maximizes the expected return, defined as follows:

\[
J (\tau^\mathcal{P}) = \mathbb{E}_{\tau \sim \pi_\theta (a \mid s)} \left[ \sum_{t=0}^{T} y^t r_t \right],
\]

where \( \tau^\mathcal{P} = (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T) \) is a trajectory generated by starting from \( s_0 \) (drawn from an initial state distribution) and following the policy \( \pi_\theta (a \mid s) \) afterwards (the superscript \( \mathcal{P} \) stands for Policy optimization). A discount factor \( y \in [0, 1] \) is used to ensure that the expected return is finite even if \( T \) is infinite.

In this paper, we use an open-source\(^3\) implementation of Proximal Policy Optimization (PPO) [34]. PPO uses stochastic gradient ascent by estimating the gradient of the expected return with respect to the policy parameters \( \theta \), i.e., \( \nabla_\theta J (\tau^\mathcal{P}) \). It does that using a so-called clipped surrogate objective function, that penalizes large policy updates, as follows:

\[
J^{\text{CLIP}} (\theta) = \mathbb{E}_t \left[ \min \left( \rho_t (\theta) A_t, \text{clip} (\rho_t (\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \right],
\]

where \( \rho_t (\theta) = \frac{\pi_\theta (a_t \mid s_t)}{\pi_{\text{old}} (a_t \mid s_t)} \) is the probability ratio of the policy, after and before an update, and \( \epsilon \in [0, 1] \) is a hyperparameter used for avoiding large policy updates. \( A_t \) is an estimation to the so-called advantage function at timestep \( t \). At each timestep, the advantage function is positive if the chosen action leads to a better reward than expected, and negative otherwise. PPO uses Generalized Advantage Estimation (GAE) [33], a simple and popular estimator for the advantage function. More information about the PPO algorithm can be found in [34].

4 METHOD

4.1 Reference Motion Generation

The first step of our approach includes automatic generation of a reference motion. For this purpose, we use FDI-MCTS [29], a recent sampling-based model-predictive algorithm for continuous control. The cost function used by FDI-MCTS uses four quadratic terms for penalizing the followings: 1) amount of torques applied to joints, 2) deviation from the default pose (shown in Fig. 4), 3) planar deviation of the center of mass from the feet mean point, and 4) the difference between the current and the target velocity of the character. We extract an approximate cycle out of the synthesized motion sequence using the method explained below.

Given a trajectory of stable locomotion movements, the cycle extraction process starts by storing the key information at every timestep. Stored information include orientation \( q^t_j \) of each joint \( j \), angular velocity \( \dot{q}^t_j \) of each bone \( b \), position \( p^t_e \) and linear velocity \( v^t_e \) of each end-effector \( e \) (e.g., hands and feet for humanoid characters), and the character’s center of mass \( p^t_{com} \). At each timestep, in order to detect the end of the cycle, positions and linear velocities of all end-effectors are compared with their corresponding values at the initial timestep of the cycle. The cycle is completed if the end-effectors have almost the same positions as in the initial timestep and their corresponding linear velocities have acute angles between them, i.e., their dot product is positive. We set the minimum length of a cycle to 10 timesteps to avoid detecting empty cycles. This gives us an easy-to-implement and computationally cheap method for extracting cycles from synthesized movements.

4.2 Self-Imitation Learning

After synthesizing a cyclic movement, we use the PPO algorithm to find a policy for performing locomotion while imitating the reference motion. In this part, we use a training mechanism similar to DeepMimic [23]. When starting a new episode, the so-called Reference State Initialization (RSI) is used, i.e., the initial state is uniformly picked from the reference motion. An episode is terminated if a bone other that the feet is in contact with the ground, or episode length exceeds some pre-defined limit. To accelerate the training process, we employ a different Early Termination (ET) mechanism, which is explained in Section 4.4.

\(^3\)https://github.com/openai/baselines
4.3 Reward Function

Our reward definition is almost identical to DeepMimic [23]. The instantaneous reward $r_t$ at timestep $t$ is defined as follows:

$$r_t = w^I r^I_t + w^T r^T_t,$$

where $r^I$ (weighted by $w^I$) and $r^T$ (weighted by $w^T$) define the imitation and task rewards at timestep $t$, respectively. This encourages the character to satisfy the task objective while imitating the reference motion. All quantities on the right side of the equation are between 0 and 1, and $w^I + w^T = 1$, leading to a simple reward $0 \leq r_t \leq 1$. This reward is also informative for humans in the sense that $r_t = 1$ means that the character’s performance is ideal and $r_t = 0$ means that the character is failing in collecting any imitation or task reward.

4.3.1 Imitation Reward. The imitation reward encourages the character to imitate the reference motion, and it is computed as a weighted sum of four terms as follows:

$$r^I_t = \alpha^I \bar{v}^I_t + w^I \bar{v}^I_t + w^I \bar{v}^I_t + w^I \bar{v}^I_t,$$

$$w^I = 0.65, \quad \bar{v}^I = 0.1, \quad \bar{v}^I = 0.15, \quad \bar{v}^I = 0.1$$

The terms $r^I_t, \bar{v}^I_t, \bar{v}^I_t,$ and $\bar{v}^I_t$ are computed exactly the same as in [23].

4.3.2 Task Reward. In this paper, we only consider the locomotion tasks. So the task reward encourages the character to walk in the desired direction with the desired speed, and it is defined as follows:

$$r^T_t = \exp \left(-2.5 \times \|\bar{v}^I - \bar{v}\|^2\right),$$

where $\bar{v}^I$ is the desired velocity and $\bar{v}$ is the mean velocity of character’s bones, projected on the $xy$-plane.

4.4 Termination Curriculum

When using RL algorithms, a common challenge appears in the initial stages of the training, where the policy can easily lead the agent to the fruitless regions of the state space. Plus, it has been recently shown that PPO is prone to getting stuck in local optima [11]. These can cause a huge waste of simulation budget during the training. We use a didactic example, shown in Fig. 3 to demonstrate this problem and how we propose to solve it.

Fig. 3b. shows how the simulation budget can be wasted even when Reference State Initialization (RSI) is used without any termination mechanisms. The figure shows a 2D state space, in which the optimal state trajectory (i.e., the reference motion) is shown in dark gray. The light-to-dark heatmap shows the state-dependent reward distribution and each solid black line resembles a random trajectory (i.e., episode) in the state space. As it can be seen in the figure, RSI forces the episodes to fork the optimal state trajectory in the beginning. However, a non-optimal policy leads to the regions of the state space that should not be visited using an optimal policy.

To mitigate the problem shown in Fig. 3b, we propose Termination Curriculum (TC), a continuous curriculum learning mechanism by limiting the minimum allowed amount of instantaneous reward in each timestep, and lowering the limit during training. This simple mechanism can lead to a significant increase of performance in terms of both reward and training speed. Next we explain how TC works in detail.

Let $R^{min}_t$ denote the threshold for the instantaneous reward at timestep $t$. We add an extra termination condition to the underlying MDP such that an episode is terminated if $r_t < R^{min}_t$. This forces the agent to only visit the regions of the state space in which $r_t \geq R^{min}_t$ except in the last timestep of each episode. By lowering the threshold $R^{min}_t$ during the training process, the agent is gradually allowed to visit other regions of the state space as well. In other words, the character performs in a restricted state space, which in the beginning is significantly smaller than the original space. By gradually relaxing the restrictions, the state space becomes larger, allowing the character to learn how to act in the new states.

Fig. 3c demonstrates how termination curriculum mechanism helps the agent to cover the states that are in proximity to the optimal state trajectory. In this case, the trajectories are shorter. However, when used along with RSI, most of the simulation budget is spent for revising the policy in proximity of the optimal state trajectory. Furthermore, lowering the reward threshold allows the agent to visit more challenging regions of the state space, extending the policy to act in longer episodes.

The main challenge when applying termination curriculum is how to choose the right range for $R^{min}_t$. As explained in Section 4.3, our DeepMimic-style reward function is human-understandable enough, and it is always between 0 and 1. Our tests show that starting with $R^{min}_t = 0.75$ and linearly decreasing it to $R^{min}_t = 0.5$ produces good results for a wide range of characters.

5 EVALUATIONS

5.1 Implementation Details

We now explain the implementation details for producing the reported results. The source code is available on GitHub4, and examples of synthesized motions can be seen in the supplemental video 5.

Physical Simulations: We used Open Dynamics Engine (ODE)[39] for physical simulations. Simulations were done in 64 parallel threads in order to accelerate the optimization and training processes.

State Features: At each timestep $t$, the state features contain position and orientation of the root bone, angular velocity of each bone, and finally, all the joint angles. These values are concatenated into a vector and used as the feature vector $s_t$.

Action Parameterization: Action parameters are defined as the reference angles $\alpha_{ref}$ for each degree of freedom. A P-controller then converts these values to reference angular velocities, i.e., $\omega_{ref} = K_P (\alpha_{ref} - \alpha_{cur})$, where $K_P = 10$ is the P-controller’s multiplier and $\alpha_{cur}$ is the current angle.

Training: We used Tensorflow [1] for training the agent using the PPO algorithm [34]. Policy and value functions were modeled using fully-connected networks with two hidden layers of size 64 with tanh activation function. Our tests showed that it is better to use tanh in the final layer of the policy network and then interpolate

---

4[ToDo] Code will be open-sourced along with the paper’s camera-ready version.

5https://youtu.be/3l6RAynQnCs
the policy’s output using the minimum and maximum angle for each degree of freedom.

Training Parameters: The parameters used for PPO training are shown in Table 1.

| Parameter                         | Value  |
|-----------------------------------|--------|
| Clipping coefficient (\(\epsilon\)) | 0.2    |
| Number of epochs per iteration   | 25     |
| Learning rate* \([10^{-4}, 10^{-7}]\) |        |
| Training iterations              | 3500   |
| Iteration simulation budget      | 4096   |
| Batch size                       | 256    |
| Discount factor \((\gamma)\)     | 0.99   |
| GAE parameter \((\lambda)\)      | 0.95   |

* Learning rate decay was used throughout the training.

5.2 Experiments Setup

5.2.1 Characters. In order to show the flexibility of our approach, we used a variety of game characters in our simulations\(^6\). The character’s physical skeletons were modeled using 3D capsules connected via 3-DOF ball-and-socket or 1-DOF hinge joints. Finally, characters were rendered using Unity 3D game engine\(^7\). Fig. 4 shows all characters used in our work along with their skeletons. All 3D models are royalty-free assets purchased from Unity Asset Store.\(^6\)

The details of these characters in the physical simulations are shown in Table 2.

Table 2: Setup details of the simulated characters

| Property               | Wolf | Orc | Mech |
|------------------------|------|-----|------|
| Height (m)             | 1.3  | 1.6 | 2.1  |
| Mass (kg)              | 50   | 30  | 30   |
| Bones                  | 17   | 13  | 8    |
| Joints                 | 16   | 12  | 7    |
| State Dimensions       | 87   | 69  | 37   |
| Action Dimensions (DoF)| 30   | 24  | 7    |

5.2.2 Experiments. We used all characters shown in Fig. 4 for solving the locomotion task, i.e., moving in the forward direction with the target speed of 1m/s.

Reference motions: The walking cycles produced in the reference motion generation stage had usually between 20 to 40 frames (depending on the character). Thus, the maximum episode length during the training was set to 100 to ensure that each episode gives the characters enough time for repeating the reference motion at least twice.

Reward validation: In order to show the effectiveness of the reward function defined in Section 4.3, at first we trained the agents directly using the FDI-MCTS cost function converted to reward as:
where $b = 10000$ is a constant used to map the cost values into the range $[0, 1]$. Note that the reward function gradient is proportional to FDI-MCTS objective function, thereby preserving the online optimization landscape.

**Termination curriculum:** In order to show the effectiveness of the termination curriculum mechanism, we tested five different termination strategies as follows (all versions use reference state initialization):

1. **No termination:** This version does not use any termination strategies (similar to Fig. 3b).
2. **Termination curriculum:** In this version, the training begins by setting the threshold $R^\text{min}_t$ (introduced in Section 4.4) to 0.75 and then linearly decaying it to 0.5 throughout the training. The next three versions are solely introduced to demonstrate the effect of decaying the threshold and thus use constant values for the threshold $R^\text{min}_t$.
3. **Tight threshold:** This version uses the constant threshold $R^\text{min}_t = 0.75$ for episode termination. This tight threshold only allows the agent to visit the states that are "almost perfect" (similar to Fig. 3c).
4. **Medium threshold:** In this version, the constant threshold $R^\text{min}_t = 0.5$ is used to limit the visible states to those with a fairly good reward value (similar to Fig. 3d).
5. **Loose threshold:** Finally, a version with the constant threshold $R^\text{min}_t = 0.25$ was defined. This threshold does not allow the agent to visit states with very bad rewards, but does not guarantee to keep it in good states.

### 5.3 Results

Each one of the five versions introduced in Section 5.2.2 was tested in five independent runs, and the mean and standard deviation of average cost and reward were recorded. In each run, a new walking cycle was generated using FDI-MCTS and then trained using PPO algorithm for 5000 iterations. All experiments were performed by an Intel Core i7-4930K 3.40GHz CPU processor and 16GB of RAM.

Examples of synthesized locomotion movements can be seen in the supplemental video.

#### 5.3.1 FDI-MCTS is efficient in producing reference motions.

Three example cycles produced in the reference motion generation stage are shown in Fig. 1 (due to lack of space, five frames of each cycle are shown). As it can be seen in Fig. 1, the initial and final frames of the cycles are pretty similar, although not exactly the same. However, slight differences are not a problem since in the self-imitation learning stage, the agent tries to imitate the reference cycle as much as possible. It causes the agent to compensate the gap between initial and final frames of the cycle, resulting in a motion that resembles the reference cycle as much as possible. All reference motions were produced in less than five minutes of CPU time.

#### 5.3.2 The combination of FDI-MCTS and PPO is better than PPO alone.

Fig. 5 plots the FDI-MCTS cost when using PPO with DeepMimic-style reward function $r_t$, explained in Section 4.3, as opposed to the reward function $r_t^O = \frac{b - O}{b}$, which directly optimizes the FDI-MCTS objective function $O$, while at the same time avoiding very large and small rewards, which is required for PPO’s value function predictor network training. As it can be seen in the figure, the DeepMimic-style reward acts as a good proxy for optimizing the FDI-MCTS cost. On the other hand, trying to directly apply PPO to the locomotion problem without the FDI-MCTS imitation reward leads to clearly inferior results.

#### 5.3.3 Termination Curriculum improves training.

The plots in Fig. 6 show how different termination strategies work in terms of average reward. Compared to other versions, the termination curriculum shows superior performance for all characters except the wolf, where its performance is similar to the tight threshold strategy. The reason is that the wolf character, unlike orc and mech, is a quadruped and in the case of quadrupeds, a wide range of policies (including the random policies) can easily avoid the character from falling down. This results in receiving high rewards in Fig. 6a, and the curriculum has less effect. However, as it can be seen for the orc (Fig. 6b) and mech (Fig. 6c) characters, the tight threshold strategy

---

https://youtu.be/3l6RAynQnCs
We proposed an approach for constructing a policy network for synthesizing stable locomotion movements for an arbitrary given character anatomy. Our approach starts by running an online optimization method (FDI-MCTS [23]) to generate a stable locomotion gait. It then extracts a cyclic motion out of the generated movement as the reference motion. In the next stage, inspired by DeepMimic [23], proximal policy optimization (PPO) [34] is used such that the character is able to accomplish the locomotion task while imitating the reference motion. In this stage, we propose Termination Curriculum (TC), a simple continuous curriculum learning mechanism to enable rapid training of the final policy. The core idea of this mechanism is to terminate the episode if the instantaneous reward becomes lower than some threshold $R_{\text{th}}$. Decreasing this threshold during the training results in a continuous curriculum which limits the state space regions that are visible to the agent during training.

In summary, our experiments show that the proposed FDI-MCTS to PPO pipeline combines the best aspects of both algorithms. FDI-MCTS allows rapid discovery and visualization of behaviors, enabling fast reward function design iteration. FDI-MCTS also provides more flexibility in the reward design, as one can simply use quadratic cost terms without worrying about excessive reward magnitude. Once a suitable gait has been found, PPO with DeepMimic reward function can produce a stable and computationally efficient neural network policy. In contrast, using FDI-MCTS alone incurs orders of magnitude higher runtime cost due to the forward simulation, and PPO alone prevents the fast reward design iteration. In absence of the FDI-MCTS-generated reference motion, PPO also failed in optimizing our locomotion reward function.

Although our approach improves the sample complexity of DeepMimic-style learning, it still has a high sample complexity. In future work, this could be improved by using more recent state-of-the-art reinforcement learning algorithms, such as Maximum a Posteriori Policy Optimisation (MPO) [2] and Soft Actor-Critic (SAC) [10]. However, even with a more advanced RL algorithm, our proposed approach of using a trajectory optimization method for reference movement generation will probably offer faster reward function and movement style design iteration, as opposed to using RL alone.

6 DISCUSSION AND CONCLUSIONS

We proposed an approach for constructing a policy network for synthesizing stable locomotion movements for an arbitrary given character anatomy. Our approach starts by running an online optimization method (FDI-MCTS [29]) to generate a stable locomotion gait. It then extracts a cyclic motion out of the generated movement as the reference motion. In the next stage, inspired by DeepMimic [23], proximal policy optimization (PPO) [34] is used such that the character is able to accomplish the locomotion task while imitating the reference motion. In this stage, we propose Termination Curriculum (TC), a simple continuous curriculum learning mechanism to enable rapid training of the final policy. The core idea of this mechanism is to terminate the episode if the instantaneous reward function becomes lower than some threshold $R_{\text{th}}$. Decreasing this threshold during the training results in a continuous curriculum which limits the state space regions that are visible to the agent during training.

In summary, our experiments show that the proposed FDI-MCTS to PPO pipeline combines the best aspects of both algorithms. FDI-MCTS allows rapid discovery and visualization of behaviors, enabling fast reward function design iteration. FDI-MCTS also provides more flexibility in the reward design, as one can simply use quadratic cost terms without worrying about excessive reward magnitude. Once a suitable gait has been found, PPO with DeepMimic reward function can produce a stable and computationally efficient neural network policy. In contrast, using FDI-MCTS alone incurs orders of magnitude higher runtime cost due to the forward simulation, and PPO alone prevents the fast reward design iteration. In

ACKNOWLEDGMENTS

([ToDo] Acknowledgments will be added here.)

REFERENCES

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Bemi Munos, Nicolas Heess, and Martin Riedmiller. 2018. Maximum a posteriori policy optimisation. arXiv preprint arXiv:1806.06920 (2018).

[3] Amin Babadi, Kourosh Naderi, and Perttu Hämäläinen. 2018. Intelligent middle-level game control. In Proceedings of IEEE Conference on Computational Intelligence and Games (IEEE CIG). IEEE.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In Proceedings of International Conference on Machine Learning (ICML). ACM, 41–48.

[5] Glen Berseth, Cheng Xie, Paul Czernek, and Michiel van de Panne. 2018. Progressiveness reinforcement learning with distillation for multi-skilled motion control. In Proceedings of International Conference on Learning, Representations (ICLR).

[6] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakos, and Simon Colton. 2012. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games 4, 1 (2012), 1–43.

[7] 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.

[8] Thomas Geijtenbeek, Michel van de Panne, and A. Frank van der Steappen. 2013. Flexible muscle-based locomotion for bipedal creatures. ACM Transactions on Graphics (TOG) 32, 6 (2013).

[9] Antti Granqvist, Tapio Takala, Jari Takatalo, and Perttu Hämäläinen. 2018. Exaggeration of Avatar Flexibility in Virtual Reality. In Proceedings of the 2018 Annual
Figure 6: Evaluating different termination strategies. As it can be seen, termination curriculum significantly accelerates the training, allowing to train characters with various anatomies in only a few hours of CPU time.

[10] Toomas Haarmoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018).

[11] Perttu Hamäläinen, Amin Babadi, Xiaoao Ma, and Jaakko Lehtinen. 2018. PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation. arXiv preprint arXiv:1810.02541 (2018).

[12] Perttu Hamäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. 2014. Online motion synthesis using sequential monte carlo. ACM Transactions on Graphics (TOG) 33, 4 (2014), 51.

[13] Nikolaus Hansen. 2006. The CMA evolution strategy: a comparing review. In Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006). IEEE, 1615–1620.

[14] Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. ACM Transactions on Graphics (TOG) 36, 4 (2017), 42.

[15] Sumit Jain, Yuting Ye, and C Karen Liu. 2009. Optimization-based interactive motion synthesis. ACM Transactions on Graphics (TOG) 28, 1 (2009), 10.

[16] Timothy F Lillcrap, 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).

[17] Jialin Liu, Diego Pérez-Liébana, and Simon M Lucas. 2016. Rolling horizon coevolutionary planning for two-player video games. In Computer Science and Electronic Engineering (CIEEE), 2016 8th. IEEE, 174–179.

[18] Libin Liu and Jessica Hodgins. 2018. Learning basketball dribbling skills using deep reinforcement learning with a stochastic actor-critic. arXiv preprint arXiv:1801.01290 (2018).

[19] Yishuang Liu, Kan Xia, Jingbo Han, and Victor Chelmoff. 2018. Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics (TOG) 37, 4 (2018), 41.

[20] Joose Rajamäki and Perttu Hamäläinen. 2017. Augmenting sampling based controllers with machine learning. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM, 11.

[21] John Schulman, Pieter Abbeel, and Sergey Levine. 2014. Rolling horizon methods for games with continuous states and actions. In Computational Intelligence and Games (CIG), 2014 IEEE Conference on. IEEE, 1–8.

[22] John Schulman, Pieter Abbeel, and Sergey Levine. 2015. Trust region policy optimization. In Proceedings of International Conference on Machine Learning (ICML). 1889–1897.

[23] Ting Lin, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.05864 (2017).

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017).

[25] Chiara F Sironi, Jialin Liu, Diego Perez-Liebana, and Simon M Lucas. 2018. Self-adaptive mcts for general video game playing. In International Conference on the Applications of Evolutionary Computation. Springer, 358–375.

[26] Chaara F Sironi, Jialin Liu, Diego Perez-Liebana, Raluca D Gaina, Ivan Bravi, Simon M Lucas, and Mark HM Winands. 2018. Self-adaptive mcts for general video game playing. In International Conference on the Applications of Evolutionary Computation. Springer, 358–375.

[27] Russel L Smith. 2001. Open Dynamics Engine. http://www.ode.org/. Accessed: 2019-01-01.

[28] Richard S Sutton and Andrew G Barto. 2018. Reinforcement learning: An introduction. MIT press.

[29] Wenhao Yu, Greg Turk, and C Karen Liu. 2018. Learning symmetric and low-energy locomotion. ACM Transactions on Graphics (TOG) 37, 4 (2018), 144.

[30] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. ACM Transactions on Graphics (TOG) 37, 4 (2018), 145.