Physics-Based Motion Imitation with Adversarial Differential Discriminators

ZIYU ZHANG*, Simon Fraser University, Canada
SERGEY BASHKIROV*, Sony Playstation, USA
DUN YANG, Simon Fraser University, Canada
YI SHI, Simon Fraser University, Canada
MICHAEL TAYLOR, Sony Playstation, USA
XUE BIN PENG, Simon Fraser University, Canada and Nvidia, Canada

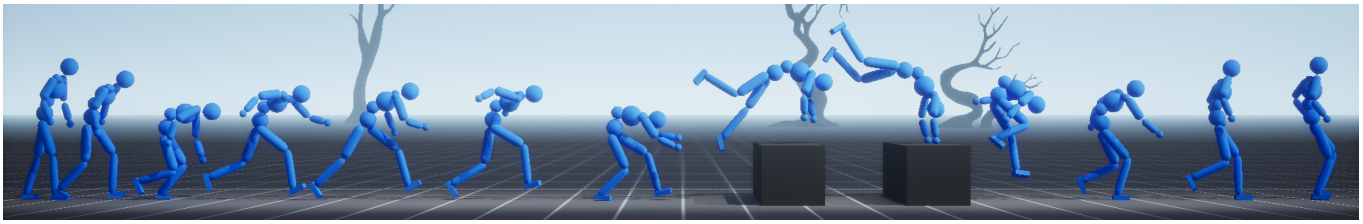


Fig. 1. We propose an adversarial multi-objective optimization technique that enables physically simulated characters to closely imitate a broad range of highly agile and athletic skills, without requiring manual reward engineering. Here, a physically simulated character learns to jump over obstacles by imitating a *double kong* reference motion.

Multi-objective optimization problems, which require the simultaneous optimization of multiple objectives, are prevalent across numerous applications. Existing multi-objective optimization methods often rely on manually-tuned aggregation functions to formulate a joint optimization objective. The performance of such hand-tuned methods is heavily dependent on careful weight selection, a time-consuming and laborious process. These limitations also arise in the setting of reinforcement-learning-based motion tracking methods for physically simulated characters, where intricately crafted reward functions are typically used to achieve high-fidelity results. Such solutions not only require domain expertise and significant manual tuning, but also limit the applicability of the resulting reward function across diverse skills. To bridge this gap, we present a novel adversarial multi-objective optimization technique that is broadly applicable to a range of multi-objective reinforcement-learning tasks, including motion tracking. Our proposed Adversarial Differential Discriminator (ADD) receives a single positive sample, yet is still effective at guiding the optimization process. We demonstrate that our technique can enable characters to closely replicate a variety of acrobatic and agile behaviors, achieving comparable quality to state-of-the-art motion-tracking methods, without relying on manually-designed reward functions.

*Joint First Authors.

Authors' Contact Information: Ziyu Zhang, zza333@sfu.ca, Simon Fraser University, Canada; Sergey Bashkirov, sergey.bashkirov@sony.com, Sony Playstation, USA; Dun Yang, dya66@sfu.ca, Simon Fraser University, Canada; Yi Shi, ysa273@sfu.ca, Simon Fraser University, Canada; Michael Taylor, mike.taylor@sony.com, Sony Playstation, USA; Xue Bin Peng, xbpeng@sfu.ca, Simon Fraser University, Canada and Nvidia, Canada.

SA Conference Papers '25, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3763819

CCS Concepts: • **Computing methodologies** → **Procedural animation**; **Adversarial learning**; *Control methods.*

Additional Key Words and Phrases: character animation, reinforcement learning, adversarial imitation learning

ACM Reference Format:
Ziyu Zhang, Sergey Bashkirov, Dun Yang, Yi Shi, Michael Taylor, and Xue Bin Peng. 2025. Physics-Based Motion Imitation with Adversarial Differential Discriminators. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3757377.3763819

1 Introduction

Physics-based character animation has seen rapid progress over the past several years. Data-driven reinforcement learning methods have enabled the field to grow from synthesizing controllers for relatively simple and common behaviors, such as locomotion, to controllers that can replicate a wide range of highly dynamic and complex skills. The effectiveness of RL methods is heavily dependent on the reward function, which tends to require significant manual effort to design and tune. As an alternative to manual reward engineering, adversarial imitation learning techniques automatically learn reward functions from data, in the form of an adversarial discriminator. Despite these impressive advances, a gap in quality has persisted between physically simulated animation and human motion capture. Prior adversarial imitation learning methods, though effective at producing natural and life-like behaviors, often diverge from the exact reference motions they aim to reproduce. Rather than focusing on precise replication of target motions, these methods capture the overall *style* of the motions via a distribution matching objective. However, closely replicating a target motion can be vital for many animation applications. In this work, we present a novel adversarial multi-objective optimization (MOO) technique, which

can be broadly applied to a wide range of MOO problems. When applied to motion tracking for character animation, our technique enables physics-based controllers to accurately mimic challenging reference motions recorded from human actors, without relying on manually engineered reward functions.

A commonly used technique for solving MOO problems is the loss balancing method, which combines multiple objectives into a scalar function via a weighted sum. Although simple, this approach heavily relies on careful weight selection, a process that can be time-consuming and labor-intensive when done manually. To address this limitation, our technique automatically aggregates multiple objectives by replacing the traditional weighted sum of objective functions with an *adversarial differential discriminator* (ADD). The discriminator takes as input a vector of objective values, herein referred to as a *differential* vector, since the objective values represent the difference between the performance of a model and the ideal performance for each objective. During training, the discriminator learns to classify whether a given differential vector corresponds to an ideal solution or not. This design enables the discriminator to automatically and dynamically determine how to balance the various objectives over the course of training, automatically honing in on more challenging objectives as the model's performance improves. Furthermore, with our formulation, the only positive sample provided to the discriminator is a zero vector representing the differential vector of an ideal "zero-error" solution. A key finding of this work is that a discriminator trained with only a single positive sample remains effective in guiding the optimization process.

The central contribution of this paper is a novel GAN-based framework for multi-objective optimization. Our framework provides an automatic and dynamic way to aggregate different objectives in MOO problems. Furthermore, unlike the traditional loss weighting approach, our framework can capture potential nonlinear relationships among the objectives. We demonstrate that our approach achieves performance comparable to existing methods that use hand-crafted objective aggregation functions across several MOO problems, including motion tracking for simulated character and non-motion-imitation tasks. Our framework successfully enables a simulated humanoid and a simulated robot to replicate a variety of highly agile and acrobatic skills, achieving quality on par with the state-of-the-art motion tracking methods, while alleviating the need for manual reward engineering.

2 Related Work

Physics-based character animation enables procedural methods that can automatically synthesize realistic and responsive behaviors for virtual characters. A key challenge has been developing controllers that can reproduce the vast array of motor skills exhibited by humans and animals, while also producing life-like movements. Early efforts leveraged human insight to design skill-specific control strategies [Coros et al. 2010; Hodgins et al. 1995; Wooten 1998; Yin et al. 2007]. These manually-designed controllers have been effective in replicating a large variety of complex motor skills [Al Borno et al. 2013; Da Silva et al. 2008; Geyer et al. 2003; Mordatch et al. 2012]. However, designing skill-specific controllers often entails a lengthy development process, which can be difficult to apply to skills where domain expertise is scarce. Optimization-based methods can mitigate the reliance on manual engineering [Al Borno et al. 2013; de Lasa et al. 2010; Mordatch et al. 2012; Naderi et al. 2017; van de Panne et al. 1994; Wampler et al. 2014], but may nonetheless require carefully-designed control structures that expose a compact set of optimization parameters. Furthermore, designing suitable objective functions that lead to naturalistic behaviors for a particular skill can present a daunting challenge [Geijtenbeek et al. 2013; Wang et al. 2009]. A recent line of work explores leveraging large language models (LLMs) to automate the design of objective functions [Cui et al. 2025; Ma et al. 2023].

Data-driven methods: Meanwhile, data-driven techniques alleviate some challenges of controller engineering by imitating reference motion data, such as motion clips acquired through motion capture or artist-animated keyframes [Sharon and van de Panne 2005; Zordan and Hodgins 2002]. One of the most common approaches for imitating motion data is via motion tracking, where a controller imitates a desired behavior by explicitly tracking target poses prescribed by a reference motion clip [Liu et al. 2005, 2010; Sok et al. 2007]. These tracking-based methods can reproduce a wide range of behaviors without designing skill-specific objectives for every behavior [Lee et al. 2010; Liu and Hodgins 2018; Liu et al. 2016]. Motion tracking combined with deep reinforcement learning has led to general frameworks that can imitate diverse motor skills [Peng et al. 2018], with systems that can reproduce hundreds of distinct behaviors using the same learning algorithm [Wang et al. 2020; Won et al. 2020; Yuan et al. 2021]. The vital component to the success of motion tracking methods lies in designing sufficiently general tracking objectives that can be applied to a large variety of skills, while also producing high-quality results for each specific skill [Ma et al. 2021; Wang et al. 2020]. Constructing general-purpose tracking objectives can therefore entail a tedious design and tuning process.

Adversarial imitation learning: Adversarial imitation learning and related inverse reinforcement learning methods provide an alternative to the manual design of objective functions by jointly learning an objective function and a corresponding policy from data through an adversarial mini-max game [Abbeel and Ng 2004; Ziebart et al. 2008]. Adversarial imitation learning can be instantiated through a GAN-like framework [Goodfellow et al. 2014], where a discriminator is trained to differentiate between behaviors from a demonstrator and behaviors produced by an agent. The agent then aims to learn a policy that produces behaviors that maximize the prediction error of the discriminator [Ho and Ermon 2016]. It has been shown that this adversarial framework imitates demonstrations by optimizing a variational approximation of the divergence between the demonstrations and the agent's behavior distribution [Ke et al. 2021; Nowozin et al. 2016]. Peng et al. [2021] leveraged adversarial imitation learning to train controllers that are able to imitate task-relevant behaviors from unstructured motion datasets containing diverse motion clips. These adversarial techniques have been able to produce high-quality motions that are comparable to state-of-the-art motion tracking methods [Peng et al. 2022, 2019; Xu and Karamouzas 2021]. However, prior adversarial imitation learning methods for motion imitation leverage a general distribution matching formulation, where an agent is only required to match

the overall motion distribution of the dataset [Peng et al. 2021; Xu and Karamouzas 2021]. This provides the agent with the flexibility to sequence motion transitions and segments in different orders, potentially dropping modes in the target motion distribution. This flexibility can be an advantage in certain applications, but it can also be detrimental in animation applications that require precise replication of a motion clip, such as in-betweening and post-processing of keyframe animations.

Multi-objective optimization: MOO problems involve optimizing multiple objectives simultaneously. A common strategy is to combine the objectives via a weighted sum. However, the effectiveness of such methods is highly sensitive to the chosen weights, which entail meticulous manual tuning. Multi-objective evolutionary algorithms can approximate the Pareto front by evolving a population of solutions [Deb et al. 2002; Xu et al. 2020]. For instance, Agrawal and van de Panne [2013] adapted $(1+\lambda)$ CMA-ES to synthesize jumping controllers with optimal trade-offs between effort and jump height [Igel et al. 2007]. However, evolutionary algorithms are computationally expensive as they need to evaluate and maintain a large population of solutions. Chen et al. [2019] proposed a more efficient method that trains a single meta-policy that can be quickly adapted to different preference vectors. Yet, it requires manual specification of preferences that can be difficult due to varying objective scales. Prior work, such as Abdolmaleki et al. [2020] and Xu et al. [2023], addresses this by introducing scale-invariant multi-critic RL frameworks, in which each objective is assigned its own critic and a single policy is jointly optimized over this set of critics. In this work, we propose an adversarial MOO approach that automatically balances the different objectives without manual preference specification or multiple critics. Moreover, our method enables nonlinear combinations of objectives that can better capture complex relationships among disparate objectives and the potentially non-convex Pareto front.

3 Background

In this work, we propose a multi-objective optimization method using an adversarial differential discriminator (ADD). To evaluate its effectiveness, we apply ADD to solve a range of control problems modeled as MDPs, where an agent aims to optimize multiple objectives concurrently. An MDP is defined as $M = (\mathcal{S}, \mathcal{A}, \gamma, \rho_0, \rho, r)$, with a state space $\mathcal{S}$, action space $\mathcal{A}$, discount factor $\gamma \in [0, 1]$, initial state distribution $\rho_0(\mathbf{s})$, dynamics function $\rho(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and reward function r. An agent interacts with the environment by sampling an action from a policy $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$ at each timestep t conditioned on state $\mathbf{s}_t$. The agent then performs the action, resulting in a new state $\mathbf{s}_{t+1}$, sampled according to the environment dynamics $\mathbf{s}_{t+1} \sim \rho(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. A trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T)$ consists of a sequence of state $\mathbf{s}_t \in \mathcal{S}$ and action $\mathbf{a}_t \in \mathcal{A}$ pairs. The goal of the agent is to learn a policy that maximizes the expected discounted return $J(\pi)$,

$$J(\pi) = \mathbb{E}_{p(\tau|\pi)}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \tag{1}$$

where $p(\tau|\pi) = \rho_0(\mathbf{s}_0)\prod_{t=0}^{T-1}\rho(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t)$ represents the likelihood of a trajectory τ under a policy π.

4 Adversarial Differential Discriminator

In MOO problems, multiple objectives are typically expressed as loss functions, where the goal is to jointly minimize these losses. Let $l^i(\cdot)$ denote the i-th loss function for $1 \le i \le n$. In the following discussion, we assume that each loss function is non-negative $l^i(\cdot) \ge 0$. A common approach for tackling MOO problems is to aggregate the individual loss functions using a weighted sum:

$$\min_\theta \sum_i w^i l^i(\theta), \tag{2}$$

where w^i denotes the corresponding weight assigned to the i-th loss function. The goal of the optimization problem then is to find a set of model parameters θ that minimizes this linear combination of loss functions [Kendall et al. 2017; Liu et al. 2019].

This formulation, however, restricts the aggregation to linear combinations of the individual objectives. In this work, we propose constructing a nonlinear aggregation using an *adversarial differential discriminator* $D(\Delta)$, which enables our method to automatically learn more flexible combinations of the objectives. In our framework, the loss functions $l^i(\theta)$ are assembled into a *differential* vector $\Delta = \left(l^1(\theta), ..., l^n(\theta)\right)$. The differential vector can be interpreted as the error, or the difference between the ideal and actual performance, for each objective. The discriminator $D(\Delta)$ acts as a nonlinear aggregation function that combines the individual losses in Δ together into a single aggregate loss. The multiple objectives are then jointly optimized through an adversarial framework [Goodfellow et al. 2014], which formulates the MOO problem as a mini-max game:

$$\min_\theta \max_D \quad \log(D(\mathbf{0})) + \log(1 - D(\Delta)). \tag{3}$$

A key distinction between ADD and prior adversarial learning frameworks is that the adversarial differential discriminator D only receives a *single* positive sample, a zero vector $\Delta = \mathbf{0}$ corresponding to the differential vector of an ideal solution. One of the key findings of this work is that an adversarial discriminator trained on a single positive sample can still be effective at solving a wide range of tasks.

However, simply optimizing Eq. 3 can lead to degenerate behaviors, where the discriminator D may converge to a delta function that assigns a score of 1 to the zero differential vector and a score of 0 to any non-zero differential vector. This delta function can lead to uninformative gradients, which can impair the optimization process. To mitigate this degeneracy, we follow Peng et al. [2021] and introduce a gradient penalty (GP) regularizer:

$$\min_\theta \max_D \quad \log(D(\mathbf{0})) + \log(1 - D(\Delta)) - \lambda^{GP}\mathcal{L}^{GP}(D), \tag{4}$$

where the gradient penalty $\mathcal{L}^{GP}$ is given by:

$$\mathcal{L}^{GP}(D) = \left\|\nabla_\phi D(\phi)\big|_{\phi=\Delta}\right\|_2^2. \tag{5}$$

When training with the objective outlined in Eq. 4, the model adjusts its parameters θ to drive Δ closer to zero to fool the discriminator. Meanwhile, the discriminator dynamically attends to different objectives and hones in on the more difficult combinations of objectives to continually challenge the model.

5 Motion Tracking with ADD

In this section, we show how ADD can be applied in a reinforcement learning framework to train control policies that enable physically

simulated characters to imitate challenging reference motions. RL-based motion tracking methods typically use a tracking reward,

$$r_t = \sum_i w^i r_t^i, \tag{6}$$

composed of a weighted sum of various reward terms r_t^i. Each reward term quantifies the error between the agent's motion and the reference motion for a specific motion feature [Chentanez et al. 2018; Peng et al. 2018; Wang et al. 2020], such as joint rotations, root positions, etc. A commonly used formulation for these reward terms is as an exponentiated error:

$$r_t^i = \exp(-\alpha^i \|\hat{\mathbf{q}}_t^i \ominus \mathbf{q}_t^i\|_2^2), \tag{7}$$

where $\mathbf{q}_t^i$ denotes a vector of features, such as position or velocities, extracted from the agent's state $\mathbf{s}_t$; $\hat{\mathbf{q}}_t^i$ are the corresponding target features specified by the reference motion; and α^i are manually specified scale parameters. Manually designing effective reward functions for precise imitation of a wide range of motions can be challenging. Moreover, the reward parameters w^i and α^i can be laborious to tune, and may need to be adjusted for different types of motions.

We propose an adaptive motion tracking reward function that models motion tracking as an MOO problem and replaces the linear weighted sum in Eq. 6 with a learned adversarial differential discriminator. During training, the discriminator $D(\Delta)$ receives the difference between the agent's state $\mathbf{s}$ and the reference motion $\hat{\mathbf{s}}$ as negative samples (i.e., $\Delta = \hat{\mathbf{s}} \ominus \mathbf{s}$). The only positive sample provided to D is $\Delta = \mathbf{0}$, representing perfect tracking with zero tracking error. The discriminator D is trained using the following objective:

$$\max_D \quad \log(D(\mathbf{0})) + \mathbb{E}_{p(\mathbf{s}|\pi)}\left[\log(1 - D(\Delta))\right] - \lambda^{GP}\mathcal{L}^{GP}(D). \tag{8}$$

Here, the gradient penalty regularizer is specified according to

$$\mathcal{L}^{GP}(D) = \mathbb{E}_{p(\mathbf{s}|\pi)}\left[\|\nabla_\phi D(\phi)|_{\phi=\Delta}\|_2^2\right], \tag{9}$$

where $p(\mathbf{s}|\pi)$ represents the marginal state distribution under the policy π. In our framework, the gradient penalty is applied to the negative samples instead of the positive samples, as ADD receives only one positive training sample. This differs from prior work, such as Peng et al. [2021], where the gradient penalty is applied exclusively to the positive samples. The reward for training the tracking policy π is then given by:

$$r_t = -\log(1 - D(\Delta_t)), \tag{10}$$

where the differential vector is simply the difference between the agent's state and the target state $\Delta_t = \hat{\mathbf{s}}_t \ominus \mathbf{s}_t$.

Previous adversarial imitation learning techniques adopt a distribution matching approach, where the discriminator $D(s_{t-n:t})$ classifies a sequence of n states as either reference or policy-generated. This encourages the policy to produce trajectories that broadly resemble the characteristics of the reference motion. In contrast, our formulation allows for precise frame-level replication, which is essential for applications requiring high accuracy, such as motion in-betweening or animation keyframe post-processing.

5.1 Discriminator Observations

Following Peng et al. [2021], an observation map $\phi(\cdot)$ extracts features from the agent's state $\mathbf{s}$ and the reference motion $\hat{\mathbf{s}}$. The differential vector then consists of the differences between the extracted features $\Delta_t = \phi(\hat{\mathbf{s}}_t) \ominus \phi(\mathbf{s}_t)$. The observation map $\phi(\cdot)$ extracts a set of features similarly to those from Peng et al. [2018]:

- Global position and rotation of the root
- Position of each joint represented in the character's local coordinate frame
- Global rotation of each joint
- Linear and angular velocity of the root represented in the character's local coordinate frame
- Local velocity of each joint

where the character's local coordinate frame is specified with the origin located at the root of the character (i.e., pelvis). The x-axis of the local coordinate frame aligns with the root link's facing direction, while the positive y-axis points in the global upward direction.

6 Motion Tracking

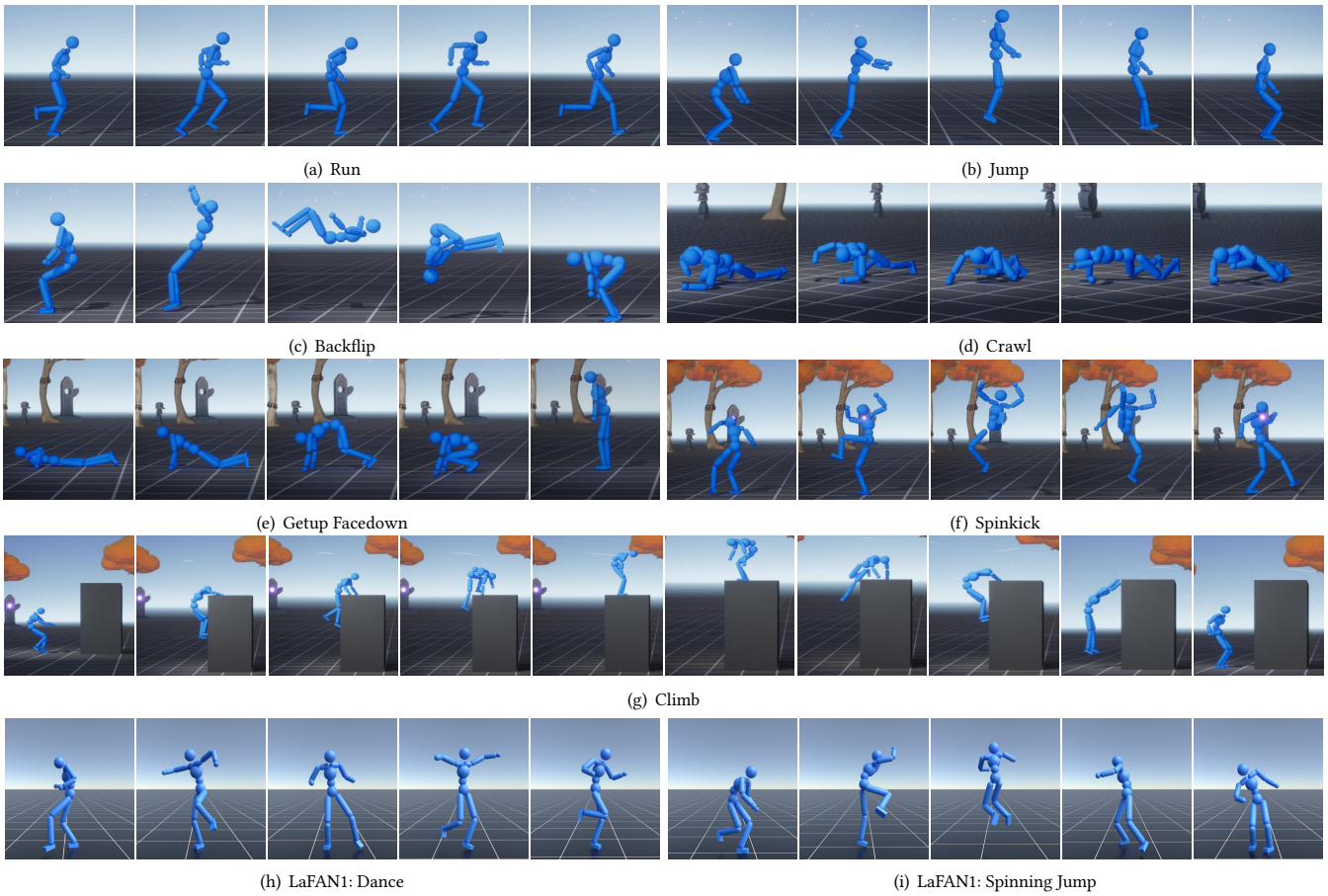
To evaluate the effectiveness of ADD for motion imitation, we apply ADD to train a 28 DoF simulated humanoid and a 26 DoF simulated Sony EVAL robot [Taylor et al. 2021] to imitate a diverse suite of motion clips. We evaluate ADD on imitating individual motion clips from Peng et al. [2018], as well as training a single general policy on larger motion datasets, such as the DanceDB subset of AMASS and a subset of LaFAN1 [Harvey et al. 2020; Mahmood et al. 2019]. The LaFAN1 subset is curated by excluding motions involving object and terrain interactions, which are not simulated in our environments. The LaFAN1 subset contains over an hour of various locomotion skills, including jumping, sprinting, fighting, dancing, etc.

6.1 States and Actions

The state $\mathbf{s}_t$ is composed of features, similar to those used by Peng et al. [2021], including the positions of each body link relative to the root, the rotations of the links encoded in the 6D normal-tangent representation, as well as the linear and angular velocities of each link. All features are recorded in the character's local coordinate frame. Target poses from the reference motion are also provided to the policy to synchronize the simulated character with the reference motion. The policy's actions $\mathbf{a}_t$ specify target rotations for each joint, which are actuated using PD controllers. Spherical joint targets are represented using 3D exponential maps [Grassia 1998], while revolute joints are represented using scalar rotation angles.

6.2 Network Architecture

The policy π is modeled with a neural network that maps a given state $\mathbf{s}_t$ to a Gaussian distribution over the actions, $\pi(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(\mu(\mathbf{s}_t), \Sigma)$. The covariance matrix Σ is fixed over the course of training, and it is represented by a diagonal matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, ...)$ with manually specified values. The input-dependent mean $\mu(\mathbf{s}_t)$ is modeled by a fully connected network with two hidden layers, consisting of 1024 and 512 ReLU-activated units [Nair and Hinton 2010], followed by a linear output layer. Similar architectures are adopted for the value function $V(\mathbf{s}_t)$ and discriminator $D(\Delta)$, except that their output layers consist of a single linear unit.



(a) Run (b) Jump

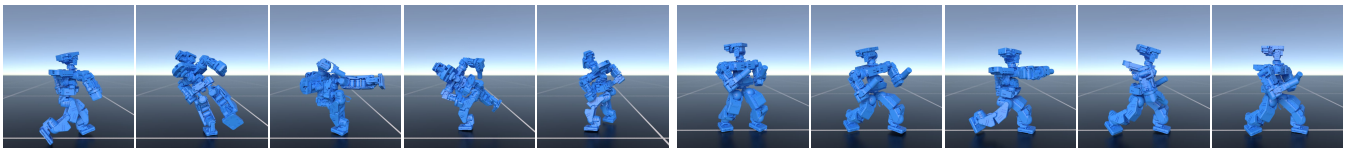
(c) Backflip (d) Crawl

(e) Getup Facedown (f) Spinkick

(g) Climb

(h) LaFAN1: Dance (i) LaFAN1: Spinning Jump

Fig. 2. Snapshots of the simulated humanoid characters trained using ADD performing various skills. ADD enables characters to replicate a diverse repertoire of behaviors, achieving tracking quality comparable to state-of-the-art motion imitation methods, without manual reward engineering.



(a) Kick (b) Punch

Fig. 3. Visual snapshots of the simulated EVAL robot replicating a range of target motions, using motion-tracking controllers trained with ADD. The controllers, trained using learned tracking rewards, successfully enable the robot to reproduce a set of challenging skills.

6.3 Training

An overview of the ADD training procedure is outlined in Algorithm 1. Following Peng et al. [2018], the character is initialized to starting states randomly sampled from the reference motion. At each timestep t, features are extracted from the agent's state $\phi(\mathbf{s}_t)$ and the target state from the reference motion $\phi(\hat{\mathbf{s}}_t)$. The difference $\Delta_t = \phi(\hat{\mathbf{s}}_t) \ominus \phi(\mathbf{s}_t)$ is then provided as input to the discriminator. The discriminator $D(\Delta_t)$ then outputs a score, which is used to calculate the reward for the agent according to Eq. 10.

The trajectories collected by the agent are recorded in an experience buffer $\mathcal{B}$. After a batch of data is collected, mini-batches are sampled from the buffer to update the discriminator, policy, and value function. The policy is updated using PPO [Schulman et al. 2017], with advantages computed with GAE(λ) [Schulman et al. 2018]. The value function is updated using target values computed with TD(λ) [Sutton and Barto 2018], and the discriminator is updated according to Eq. 8.

ALGORITHM 1: ADD Training Procedure for Motion Imitation

input $\mathcal{M}$: a reference motion clip or dataset
$D \leftarrow$ initialize discriminator
$\pi \leftarrow$ initialize policy
$V \leftarrow$ initialize value function
$\mathcal{B} \leftarrow \emptyset$ initialize experience buffer

while not done **do**
  for trajectory $i = 1, ..., m$ **do**
    $\tau^i \leftarrow \{(\mathbf{s}_t, \mathbf{a}_t)_{t=0}^{T-1}, \mathbf{s}_T\}$ collect trajectory with π
    $\hat{\tau}^i \leftarrow \{(\hat{\mathbf{s}}_t)_{t=0}^{T}\}$ sample reference trajectory from $\mathcal{M}$
    for time step $t = 0, ..., T-1$ **do**
      $\Delta_t \leftarrow \phi(\hat{\mathbf{s}}_t) \ominus \phi(\mathbf{s}_t)$
      $d_t \leftarrow D(\Delta_t)$
      $r_t \leftarrow$ calculate reward according to Equation 10 using d_t
      record r_t in τ^i
    end for
    store τ^i in $\mathcal{B}$
  end for

  for update step = 1, ..., n **do**
    $b^\pi \leftarrow$ sample batch of K differentials $\{\Delta_j\}_{j=1}^{K}$ from $\mathcal{B}$
    update D (Equation 8), V, and π using b^π
  end for
end while

6.4 Motion Imitation Results

To benchmark ADD's motion tracking performance, we compare ADD against two well-established methods: DeepMimic and AMP [Peng et al. 2018, 2021]. DeepMimic, designed for precise motion tracking, relies on a manually-designed imitation reward function composed of multiple sub-terms,

$$r_t^{\text{DM}} = w^p r_t^p + w^{jv} r_t^{jv} + w^{rv} r_t^{rv} + w^e r_t^e + w^c r_t^c. \tag{11}$$

Each term is an exponentiated error (Eq. 7), with scale α^i and weight w^i hyperparameters (see Supplementary B.2 for the full formulation). Additionally, computing the joint pose r_t^p and velocity rewards r_t^{jv} involves joint-specific weights, requiring additional tuning effort. Similar to ADD, DeepMimic policies receive phase information in the form of target frames, synchronizing the policy with a given reference motion. In contrast, AMP adopts an adversarial imitation learning framework to imitate the general style of a motion dataset, rather than exact motion tracking. While it is not designed for precise motion tracking, AMP is included in the comparisons because ADD builds upon a similar adversarial framework. However, by adapting AMP's imitation objective, our approach can change the learning objective from general style imitation to accurate motion tracking. Many follow-up works extend DeepMimic and AMP with architectural innovations orthogonal to the imitation objective [Luo et al. 2023; Peng et al. 2022; Tessler et al. 2024]. However, our experiments focus on the core imitation objective. Therefore, we compare our method directly with DeepMimic and AMP. ADD can also be combined with these additional enhancements in a similar way that DeepMimic and AMP have been extended.

For a fair comparison across methods, we disable pose termination used in Peng et al. [2018], which terminates an episode if the character's pose deviates significantly from the reference. Pose termination is not applicable to distribution matching techniques such as AMP, where the policy is not synchronized with the reference motion. Early termination is triggered only when the character makes undesired contact with the ground. The baselines are implemented based on publicly available code provided and tuned by Peng et al. [2018, 2021] to ensure reliable comparisons. Motion tracking performance is evaluated using the position tracking error e_t^{pos}, and DoF velocity tracking error, which is an indicator of motion smoothness. e_t^{pos} measures the difference in the root position and relative joint positions between the simulated character and the reference motion:

$$e_t^{\text{pos}} = \frac{1}{N^{\text{joint}} + 1}\Bigg(\sum_{j \in \text{joints}} \left\| (\hat{\mathbf{x}}_t^j - \hat{\mathbf{x}}_t^{\text{root}}) - (\mathbf{x}_t^j - \mathbf{x}_t^{\text{root}}) \right\|_2 + \left\| \hat{\mathbf{x}}_t^{\text{root}} - \mathbf{x}_t^{\text{root}} \right\|_2 \Bigg). \tag{12}$$

Here, $\mathbf{x}_t^j$ and $\hat{\mathbf{x}}_t^j$ represent the 3D Cartesian position of joint j from the simulated character and the reference motion, respectively. N^{joint} denotes the number of joints in the character. Detailed hyperparameter settings are available in Supplementary B.4.

Figures 2 and 3 showcase behaviors learned by the humanoid and EVAL robot trained via ADD. The behaviors are best viewed in the supplementary video. ADD can closely imitate individual motion clips and larger motion datasets with different embodiments, successfully reproducing a diverse set of agile and acrobatic skills. This includes challenging parkour skills, such as Climb and Double Kong, which require particularly high motion tracking accuracy to replicate intricate contacts with the environment. A qualitative comparison of ADD against other methods is available in the supplementary video.

Table 1 and Table 2 summarize quantitative comparisons of the different methods. AMP exhibits poor tracking performance, since the policies are trained using a general distribution-matching objective. AMP's susceptibility to mode collapse also makes it less effective at tracking larger motion datasets. Both ADD and DeepMimic are able to accurately track a wide variety of reference motions. However, there are important distinctions in the robustness of the reward function and ease of deployment. The reliance on manually-designed reward functions, to a degree, limits DeepMimic's ability to effectively imitate a broad spectrum of motions, as it can be challenging to craft a general and effective reward function that can imitate a diverse variety of behaviors. For instance, DeepMimic policies failed to reproduce some of the challenging parkour motions. For the Double Kong motion, DeepMimic policies fail to jump over the boxes and learn to simply mimic running in place after tripping. In contrast, ADD successfully enables characters to clear the obstacles, replicating the jumps and intricate contacts. Policies trained using DeepMimic also exhibit notable jitteriness when tracking the DanceDB dataset, whereas ADD consistently produces smooth behaviors across diverse motions, as indicated by its lower DoF velocity tracking errors. Additional reward tuning, especially for DanceDB, may further improve DeepMimic's performance. But by automatically learning to balance different objectives instead of relying on fixed manually-specified parameters, ADD offers a more general and adaptive approach for imitating more diverse motions.

Table 1. Motion tracking performance of simulated humanoid characters trained using AMP [Peng et al. 2021], DeepMimic [Peng et al. 2018], and our method ADD. Position (Eq. 12) and DoF Velocity tracking errors are averaged ± 1 std across 5 models initialized with random seeds. Due to computational constraints, 1 model is trained for the LaFAN1 subset. For each model, errors are averaged across 4096 test episodes. ADD achieves tracking performance comparable to DeepMimic when imitating individual motion clips and larger motion datasets, while alleviating the need for manual reward engineering.

Skill	Length (s)	Position Tracking Error [m]			DoF Velocity Tracking Error [rad/s]		
		AMP	DeepMimic	ADD (ours)	AMP	DeepMimic	ADD (ours)
Run	0.80	$0.163^{\pm 0.008}$	$\mathbf{0.013^{\pm 0.002}}$	$0.165^{\pm 0.017}$	$2.811^{\pm 0.048}$	$0.584^{\pm 0.054}$	$\mathbf{0.478^{\pm 0.007}}$
Jog	0.83	$0.120^{\pm 0.007}$	$\mathbf{0.021^{\pm 0.000}}$	$0.024^{\pm 0.004}$	$2.017^{\pm 0.052}$	$0.575^{\pm 0.007}$	$\mathbf{0.507^{\pm 0.010}}$
Sideflip	2.44	$0.387^{\pm 0.011}$	$\mathbf{0.138^{\pm 0.004}}$	$0.145^{\pm 0.006}$	$2.276^{\pm 0.014}$	$\mathbf{1.118^{\pm 0.034}}$	$1.350^{\pm 0.049}$
Crawl	2.93	$0.050^{\pm 0.006}$	$\mathbf{0.027^{\pm 0.000}}$	$0.028^{\pm 0.002}$	$0.646^{\pm 0.089}$	$0.430^{\pm 0.006}$	$\mathbf{0.283^{\pm 0.002}}$
Roll	2.00	$0.141^{\pm 0.031}$	$\mathbf{0.115^{\pm 0.132}}$	$0.152^{\pm 0.005}$	$1.576^{\pm 0.318}$	$\mathbf{0.994^{\pm 0.051}}$	$1.330^{\pm 0.101}$
Double Kong	5.17	$0.214^{\pm 0.014}$	$0.223^{\pm 0.006}$	$\mathbf{0.030^{\pm 0.001}}$	$1.309^{\pm 0.043}$	$0.784^{\pm 0.013}$	$\mathbf{0.473^{\pm 0.004}}$
Getup Facedown	3.03	$0.096^{\pm 0.018}$	$0.023^{\pm 0.001}$	$\mathbf{0.022^{\pm 0.001}}$	$0.838^{\pm 0.029}$	$0.433^{\pm 0.008}$	$\mathbf{0.325^{\pm 0.005}}$
Spinkick	1.28	$0.064^{\pm 0.010}$	$0.078^{\pm 0.062}$	$\mathbf{0.025^{\pm 0.000}}$	$1.453^{\pm 0.327}$	$1.222^{\pm 0.233}$	$\mathbf{0.774^{\pm 0.007}}$
Cartwheel	2.71	$0.076^{\pm 0.006}$	$0.144^{\pm 0.153}$	$\mathbf{0.017^{\pm 0.000}}$	$0.722^{\pm 0.020}$	$0.659^{\pm 0.160}$	$\mathbf{0.317^{\pm 0.002}}$
Backflip	1.75	$0.267^{\pm 0.015}$	$0.111^{\pm 0.054}$	$\mathbf{0.062^{\pm 0.001}}$	$2.243^{\pm 0.113}$	$1.103^{\pm 0.024}$	$\mathbf{0.878^{\pm 0.013}}$
Climb	9.97	$0.185^{\pm 0.021}$	$0.066^{\pm 0.004}$	$\mathbf{0.046^{\pm 0.031}}$	$1.190^{\pm 0.012}$	$0.567^{\pm 0.015}$	$\mathbf{0.374^{\pm 0.071}}$
Dance A	1.62	$0.065^{\pm 0.009}$	$0.065^{\pm 0.029}$	$\mathbf{0.028^{\pm 0.007}}$	$0.895^{\pm 0.108}$	$0.830^{\pm 0.090}$	$\mathbf{0.428^{\pm 0.014}}$
Walk	0.96	$0.132^{\pm 0.021}$	$0.009^{\pm 0.001}$	$\mathbf{0.009^{\pm 0.001}}$	$1.394^{\pm 0.123}$	$0.286^{\pm 0.005}$	$\mathbf{0.213^{\pm 0.003}}$
AMASS DanceDB	2,804	$0.299^{\pm 0.001}$	$0.045^{\pm 0.002}$	$\mathbf{0.044^{\pm 0.001}}$	$1.156^{\pm 0.009}$	$0.504^{\pm 0.019}$	$\mathbf{0.387^{\pm 0.010}}$
LaFAN1 subset	5,470	$0.438^{\pm 0.000}$	$0.029^{\pm 0.000}$	$\mathbf{0.028^{\pm 0.000}}$	$1.302^{\pm 0.000}$	$0.511^{\pm 0.000}$	$\mathbf{0.393^{\pm 0.000}}$

Table 2. Position tracking errors (Eq. 12) of simulated EVAL robots trained using ADD, AMP [Peng et al. 2021], and DeepMimic [Peng et al. 2018]. Only the position tracking errors are reported for brevity, showing the mean ± 1 standard deviation across three random seeds, with 4096 test episodes per seed. The results show that ADD maintains strong tracking performance comparable to DeepMimic on a different character morphology.

Skill	Position Tracking Error [m]		
	AMP	DeepMimic	ADD (ours)
Walk	$0.020^{\pm 0.001}$	$\mathbf{0.013^{\pm 0.002}}$	$0.036^{\pm 0.002}$
Kick	$0.030^{\pm 0.00}$	$0.012^{\pm 0.000}$	$\mathbf{0.008^{\pm 0.000}}$
Punch	$0.020^{\pm 0.003}$	$0.007^{\pm 0.000}$	$\mathbf{0.005^{\pm 0.000}}$

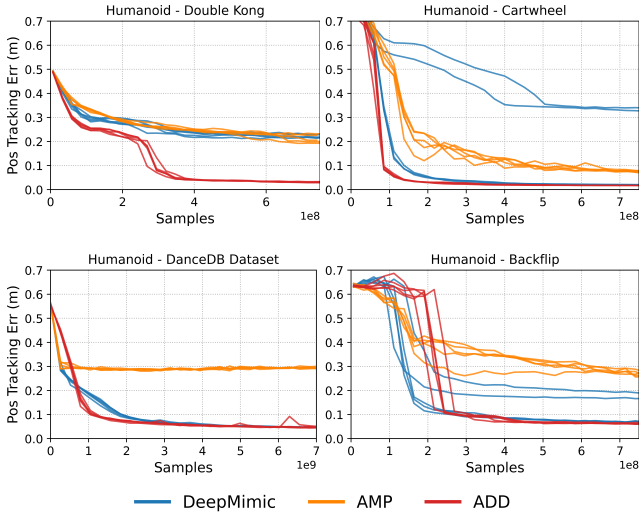


Fig. 4. Learning curves of AMP, DeepMimic, and ADD, each trained with 5 different random seeds. ADD demonstrates better consistency than DeepMimic across different seeds. DeepMimic policies converge to suboptimal behaviors in half of the seeds when tracking the Backflip and Cartwheel motions.

Figure 4 compares the learning curves of humanoid characters trained using ADD, AMP, and DeepMimic. ADD demonstrates better consistency than DeepMimic across different seeds. DeepMimic policies converge to local optima in half of the seeds when imitating the Backflip and Cartwheel motions. Learning curves for all skills and characters can be found in Supplementary B.5.

Achieving high-quality motion tracking with DeepMimic often requires careful tuning of numerous reward parameters. This tuning process can require significant domain knowledge and time. Some parameter settings might improve a subset of the objectives, but degrade the performance of other objectives. Poorly selected parameters can result in low motion quality and inferior sample efficiency, which presents a challenge for applying DeepMimic to new reference motions or characters. In our experiments, DeepMimic's hyperparameters required careful re-tuning when switching from the humanoid character to the EVAL robot to maintain good tracking performance (Supplementary B.4). In comparison, ADD can automatically weigh the different objectives, reducing the burden of manual reward engineering. The sensitivity analysis of DeepMimic in Supplementary B.3 provides further experiments that highlight

Table 3. Performance of various methods when applied to a composite task that combines motion imitation with a target steering objective. Motion imitation performance is measured via the position tracking error, while task performance is measured by the velocity error $\|\mathbf{v}_t - v^*\mathbf{d}_t^*\|$, where $\mathbf{v}_t$ is the 2D root velocity of the character and $v^*\mathbf{d}_t^*$ the 2D target velocity. ADD, via automatically balancing the different objectives, achieved optimal performance on both objectives.

	Pos Tracking Err [m]		**Target Vel Err [m/s]**	
Skill / Method	Run	Walk	Run	Walk
AMP	$0.188^{\pm 0.001}$	$0.117^{\pm 0.001}$	$0.810^{\pm 0.015}$	$0.374^{\pm 0.042}$
DeepMimic	$0.048^{\pm 0.000}$	$0.034^{\pm 0.003}$	$0.882^{\pm 0.038}$	$0.411^{\pm 0.049}$
ADD (ours)	$\mathbf{0.029}^{\pm \mathbf{0.001}}$	$\mathbf{0.024}^{\pm \mathbf{0.001}}$	$\mathbf{0.803}^{\pm \mathbf{0.009}}$	$\mathbf{0.274}^{\pm \mathbf{0.003}}$

the challenges of DeepMimic's reward design and the robustness of ADD.

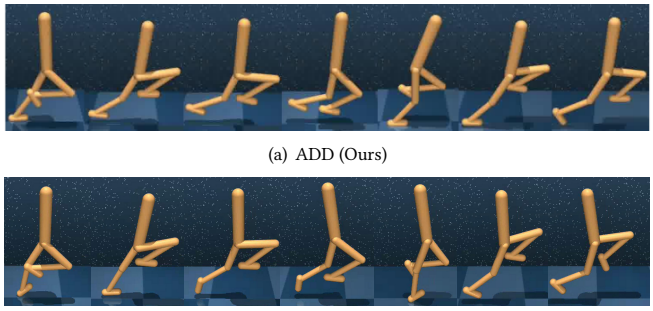
However, ADD can struggle to reproduce certain highly dynamic motions (e.g., sideflip and roll), often converging to suboptimal behaviors. In the case of the sideflip, the policy learns to stand on the ground without executing the full flip. Prior methods address this by incorporating additional early termination heuristics [Luo et al. 2023; Peng et al. 2018], which we deliberately omitted to ensure a fair comparison and focus our evaluations on the effects of different imitation objectives. Additionally, ADD is less precise than DeepMimic on some forward locomotion skills, such as humanoid running and robot walking. The performance gap stems from ADD's difficulty in accurately tracking the root position, which is crucial for imitating forward locomotion behaviors. We hypothesize that this shortcoming arises due to the root position only accounting for three elements within the high-dimensional differential vector Δ, and the gradient penalty restricts the discriminator from placing significantly greater emphasis on the root position errors.

6.5 Tasks

In this section, we evaluate the effectiveness of ADD on training control policies that can both imitate motion clips and accomplish additional task objectives. Our experiments focus on a target steering task, where the objective is to move at a target speed v^* along a target direction $\mathbf{d}_t^*$, specified by a 2D unit vector on the horizontal plane. For ADD, these task objectives are incorporated directly into the differential vector Δ, whereas for AMP and DeepMimic, a separate task reward is introduced alongside the imitation reward (Supplementary B.6). The target direction and speed are provided as observations to the policy and are randomized during training. Table 3 summarizes the performance of various methods. ADD is able to automatically balance the various objectives, enabling policies to follow the steering commands accurately while closely imitating the desired reference motion.

7 Non-Motion-Imitation Tasks

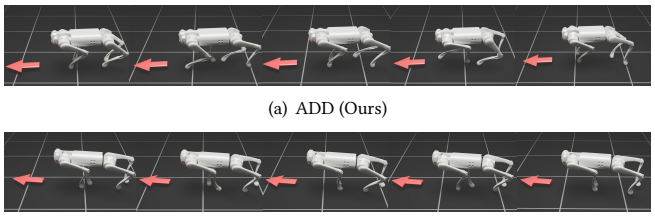
To demonstrate that ADD is effective beyond motion imitation tasks, we evaluate ADD on the Walker task [Tassa et al. 2018], a standard RL benchmark task, and a UniTree Go1 walking task from a widely-used framework for robotic locomotion [Rudin et al. 2022]. The Walker task consists of 3 different objectives, whereas the Go1 task presents a much more complex reward function with 12 objectives, including 3 dynamic steering commands. These experiments demonstrate ADD's ability to balance a number of competing objectives.



(a) ADD (Ours)

(b) Manual Reward

Fig. 5. Qualitative results of ADD on the Walker task, a standard RL benchmark. The walker trained using ADD exhibits behaviors comparable in quality to those learned with manually designed reward functions from Tassa et al. [2018].



(a) ADD (Ours)

(b) Manual Reward

Fig. 6. Qualitative results of ADD on training a UniTree Go1 quadruped to walk. The arrow denotes the steering command. Compared to controllers trained with manually tuned rewards [Rudin et al. 2022], the ADD Go1 policy displays more natural gaits, with greater foot lift and longer strides.

7.1 Training

Policies are trained using PPO with either ADD or manually-designed reward functions from Tassa et al. [2018] and Rudin et al. [2022]. The ADD training procedure largely follows Algorithm 1, with the key difference being the absence of reference motions. Instead, the target state $\hat{\mathbf{s}}_t$ is composed of target values specified by the task, such as velocity commands or target orientations. Implementation details for each task are available in Supplementary C and D.

7.2 Results

Figures 5 and 6 present qualitative comparisons between ADD and manually-designed reward functions on the two tasks: training a 2D walker to run and a quadrupedal Go1 robot to walk while following steering commands. Examples of the learned behaviors are available in the supplementary video.

As shown in Figure 5, the ADD policy produces upright and fast running behaviors of comparable performance to those produced

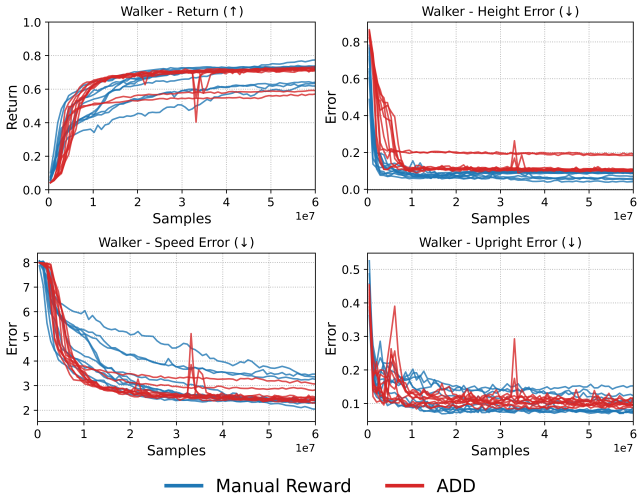


Fig. 7. Learning curves comparing ADD to the manually designed reward function from Tassa et al. [2018] on the 2D Walker task. Returns are calculated according to the reward function from Tassa et al. [2018] and normalized between the minimum and maximum possible returns per episode. Statistics are computed over 10 random seeds. ADD demonstrates comparable performance and sample efficiency to the hand-crafted reward function, despite requiring no manual reward engineering or tuning.

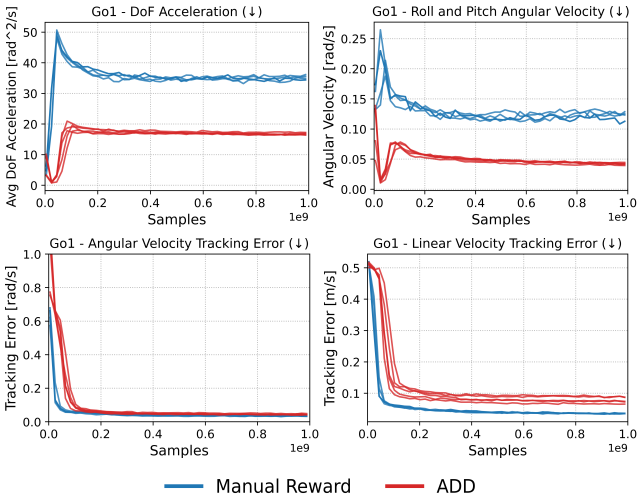


Fig. 8. Learning curves comparing ADD to the manually tuned reward function from Rudin et al. [2022] on training a Go1 quadruped to move. Results are shown across 5 random seeds per method. While ADD performs slightly worse in following linear velocity commands, it achieves lower roll and pitch angular velocities–indicating a more stable robot base–and lower DoF accelerations, which means smoother control over time.

by the manually-designed reward function. This observation is supported by the learning curves in Figure 7, which compares the two methods on the returns and the three different training objectives. ADD achieves similar final performance and sample efficiency as the manually-designed rewards. ADD produces a final return of

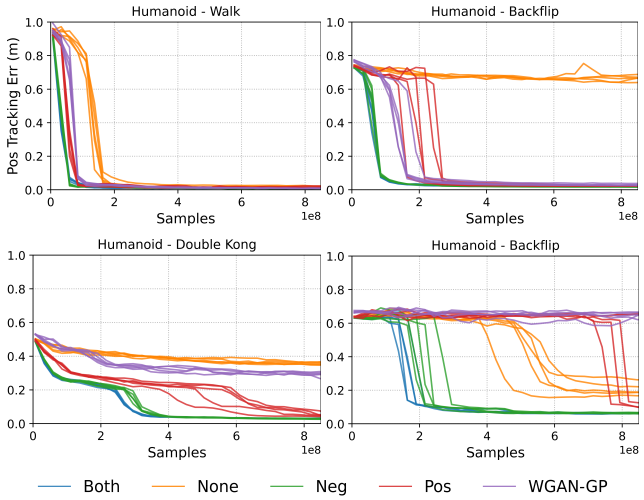


Fig. 9. Learning curves of ADD trained, under varying gradient penalty configurations, to track different reference motions. We conduct ablations over five gradient penalty configurations: none (None), penalty applied only to negative samples (Neg), only to positive samples (Pos), to both sample types (Both), and to interpolations between positive and negative samples as proposed in Gulrajani et al. [2017] (WGAN-GP). Results indicate that Neg and Both significantly outperform the other settings in tracking performance, with Neg and Both achieving comparable results. These results highlight the importance of applying the gradient penalty to negative samples for ADD.

0.691 ± 0.053, whereas the manual rewards result in a final return of 0.705 ± 0.051. Furthermore, Figure 7 shows that ADD demonstrates better consistency across training runs.

On the Go1 task, ADD enables the Go1 robot to develop more natural gaits, characterized by greater foot lift and longer strides. In comparison, the manually-designed reward produces less natural behaviors that exhibit more jittering and small shuffling steps. Figure 8, which compares ADD and Rudin et al. [2022] on key objectives, shows that while ADD performs slightly worse on following linear velocity commands, it achieves lower roll and pitch angular velocities, indicating a more stable robot base. Additionally, ADD produces smoother movements with lower DoF accelerations, a desirable property for real-world robotic deployment. A comprehensive set of learning curves for all training objectives is available in Supplementary D.3.

Across different tasks and embodiments, ADD consistently matches the performance of carefully designed reward functions with respect to performance, motion quality, consistency, and sample efficiency, all while alleviating the dependence on manual reward engineering. While manually tuned methods can be effective when well-calibrated, they often require significant domain knowledge and effort to balance different objectives. These results collectively highlight ADD's generality as an effective solution for a diverse range of multi-objective RL tasks.

8 Gradient Penalty Ablation

Gradient penalty is an effective regularizer in adversarial imitation learning methods [Mescheder et al. 2018; Peng et al. 2021]. In this section, we investigate the impact of different gradient penalty configurations on the effectiveness of ADD. We examine five configurations — no gradient penalty at all (None), gradient penalty applied exclusively to negative samples (Neg), exclusively to positive samples (Pos), to both positive and negative samples (Both), and to random interpolations between positive and negative samples as proposed in Gulrajani et al. [2017] (WGAN-GP). In the WGAN-GP setting, following Gulrajani et al. [2017], the gradient penalty is modified to enforce 1-Lipschitz continuity,

$$\mathcal{L}^{GP}(D) = \mathbb{E}_{p(\mathbf{s}|\pi)}\left[\left(\|\nabla_{\phi} D(\phi)|_{\phi=\Delta}\|_2 - 1\right)^2\right], \tag{13}$$

as opposed to Eq. 9. Figure 9 shows that models trained without regularization (None) or with WGAN-GP exhibit significantly higher tracking errors. Although the WGAN-GP has proven effective for prior adversarial methods, it does not appear to be suitable for ADD. Models trained under the Pos configuration, while notably more accurate than the None and WGAN-GP settings, result in higher tracking errors and worse sample efficiency compared to Neg and Both settings. Models trained under the Neg and Both settings produce comparable results in terms of tracking error, with the Both configuration showing a slight advantage with regard to sample efficiency. Nonetheless, these findings validate the importance of applying the gradient penalty to negative samples for the adversarial differential discriminator, instead of only positive samples as done in prior work [Peng et al. 2021]. Moreover, these results underscore the critical role of the gradient penalty in stabilizing training and enabling effective learning, as models trained without it often fail to learn effective policies.

9 Discussion and Future Work

In this work, we present an adversarial multi-objective optimization technique that is broadly applicable to a variety of tasks, including motion imitation for physics-based character animation. Our technique enables both a simulated humanoid and a simulated EVAL robot to accurately replicate a diverse suite of challenging skills, achieving comparable performance to state-of-the-art motion tracking methods, without requiring manual reward engineering. Despite its effectiveness, our method has certain limitations. One major limitation is that ADD is susceptible to converging to locally optimal behaviors. For example, when attempting to reproduce the highly dynamic rolling motion, characters trained with ADD learn to simply lie down and get up, rather than completing a full roll. Prior motion tracking methods typically mitigate the chances of converging to such behaviors by incorporating additional heuristics, such as early termination based on tracking error [Luo et al. 2023; Peng et al. 2018]. Furthermore, recent studies show that training controllers on massive motion datasets is a scalable approach to enable physically simulated characters to acquire diverse behaviors [Luo et al. 2023; Tessler et al. 2024; Yao et al. 2024]. We would like to explore, in future work, applying ADD to train a general controller that does not rely on hand-crafted reward functions, a common dependency in existing frameworks. Finally, another interesting avenue for future work is to apply ADD to multi-objective optimization or multi-task problems in other domains, such as computer vision and natural language processing.

Acknowledgments

We would like to express our gratitude to Michael Xu for the help with data processing, Beyond Capture Studios for providing the high-quality parkour motion data, and Yvette Chen for the 3D artwork that elevated the overall visual presentation of our work. This work was supported by NSERC via a Discovery Grant (RGPIN-2015-04843).

References

Pieter Abbeel and Andrew Y. Ng. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 1. https://doi.org/10.1145/1015330.1015430

Abbas Abdolmaleki, Sandy H. Huang, Leonard Hasenclever, Michael Neunert, H. Francis Song, Martina Zambelli, Murilo F. Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. 2020. A distributional view on multi-objective policy optimization. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 2, 12 pages.

Shailen Agrawal and Michiel van de Panne. 2013. Pareto Optimal Control for Natural and Supernatural Motions. In *Proceedings of Motion on Games* (Dublin 2, Ireland) *(MIG '13)*. Association for Computing Machinery, New York, NY, USA, 29–38. https://doi.org/10.1145/2522628.2522902

M. Al Borno, M. de Lasa, and A. Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics* 19, 8 (2013), 1405–1414. https://doi.org/10.1109/TVCG.2012.325

Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. 2019. Meta-Learning for Multi-objective Reinforcement Learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 977–983. https://doi.org/10.1109/IROS40897.2019.8968092

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-Based Motion Capture Imitation with Deep Reinforcement Learning *(MIG '18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/3274247.3274506

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

Jieming Cui, Tengyu Liu, Ziyu Meng, Jiale Yu, Ran Song, Wei Zhang, Yixin Zhu, and Siyuan Huang. 2025. GROVE: A Generalized Reward for Learning Open-Vocabulary Physical Skill. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

M. Da Silva, Y. Abe, and J. Popovic. 2008. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Computer Graphics Forum* (2008).

Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. 2010. Feature-Based Locomotion Controllers. *ACM Transactions on Graphics* 29, 3 (2010).

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics* 32, 6 (2013).

Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. 2003. Positive force feedback in bouncing gaits? *Proc. Royal Society of London B: Biological Sciences* 270, 1529 (2003), 2173–2183.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

F. Sebastian Grassia. 1998. Practical Parameterization of Rotations Using the Exponential Map. *J. Graphics, GPU, & Game Tools* 3 (1998), 29–48. https://api.semanticscholar.org/CorpusID:9489978

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5769–5779.

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust Motion In-Betweening. 39, 4 (2020).

Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 4565–4573.

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. 1995. Animating human athletics. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*, Susan G. Mair and Robert Cook (Eds.). ACM, 71–78. https://doi.org/10.1145/218380.218414

Christian Igel, Nikolaus Hansen, and Stefan Roth. 2007. Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary computation* 15 (02 2007), 1–28. https://doi.org/10.1162/evco.2007.15.1.1

Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. 2021. Imitation learning as f-divergence minimization. In *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*. Springer, 313–329.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2017. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2017), 7482–7491. https://api.semanticscholar.org/CorpusID:4800342

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-Driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 8 pages. https://doi.org/10.1145/1778765.1781155

C. Karen Liu, Aaron Hertzmann, and Zoran Popović. 2005. Learning Physics-Based Motion Style with Nonlinear Inverse Optimization. *ACM Trans. Graph.* 24, 3 (jul 2005), 1071–1081. https://doi.org/10.1145/1073204.1073314

Libin Liu and Jessica Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 142 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201315

Libin Liu, Michiel van de Panne, and KangKang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Transactions on Graphics* 35, 3 (2016).

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based contact-rich motion control. *ACM Trans. Graph.* 29, 4, Article 128 (July 2010), 10 pages. https://doi.org/10.1145/1778765.1778865

Shikun Liu, Edward Johns, and Andrew J. Davison. 2019. End-To-End Multi-Task Learning With Attention. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1871–1880. https://doi.org/10.1109/CVPR.2019.00197

Zhengyi Luo, Jinkun Cao, Alexander W. Winkler, Kris Kitani, and Weipeng Xu. 2023. Perpetual Humanoid Control for Real-time Simulated Avatars. In *International Conference on Computer Vision (ICCV)*.

Li-Ke Ma, Zeshi Yang, Tong Xin, Baining Guo, and KangKang Yin. 2021. Learning and Exploring Motor Skills with Spacetime Bounds. *Computer Graphics Forum* 40, 2 (2021), 251–263.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-Level Reward Design via Coding Large Language Models. *arXiv preprint arXiv: Arxiv-2310.12931* (2023).

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. 2019. AMASS: Archive of Motion Capture As Surface Shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, N. Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. *ArXiv* abs/2108.10470 (2021). https://api.semanticscholar.org/CorpusID:237277983

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. 2018. Which Training Methods for GANs do actually Converge?. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 3481–3490.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of Complex Behaviors through Contact-Invariant Optimization. *ACM Trans. Graph.* 31, 4, Article 43 (July 2012), 8 pages. https://doi.org/10.1145/2185520.2185539

Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. 2017. Discovering and Synthesizing Humanoid Climbing Movements. *ACM Trans. Graph.* 36, 4, Article 43 (jul 2017), 11 pages. https://doi.org/10.1145/3072959.3073707

Vinod Nair and Geoffrey Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. *Proceedings of ICML* 27, 807–814.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. 2016. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc., 271–279. https://proceedings.neurips.cc/paper/2016/file/cedebb6e872f539bef8c3f919874e9d7-Paper.pdf

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201311

Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. 2022. ASE: Large-scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters. *ACM Trans. Graph.* 41, 4, Article 94 (July 2022).

Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. 2019. Variational Discriminator Bottleneck: Improving Imitation Learning, Inverse RL, and GANs by Constraining Information Flow. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyxPx3R9tm

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.* 40, 4, Article 1 (July 2021), 15 pages. https://doi.org/10.1145/3450626.3459670

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. 2022. Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. In *Proceedings of the 5th Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 164)*, Aleksandra Faust, David Hsu, and Gerhard Neumann (Eds.). PMLR, 91–100. https://proceedings.mlr.press/v164/rudin22a.html

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2018. High-Dimensional Continuous Control Using Generalized Advantage Estimation. arXiv:1506.02438 [cs.LG]

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

Dana Sharon and Michiel van de Panne. 2005. Synthesis of Controllers for Stylized Planar Bipedal Walking. In *Proc. of IEEE International Conference on Robotics and Animation*.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating Biped Behaviors from Human Motion Data. *ACM Trans. Graph.* 26, 3 (July 2007), 107–es. https://doi.org/10.1145/1276377.1276511

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. 2018. DeepMind Control Suite. *arXiv e-prints*, Article arXiv:1801.00690 (Jan. 2018), arXiv:1801.00690 pages. https://doi.org/10.48550/arXiv.1801.00690 arXiv:1801.00690 [cs.AI]

Michael Taylor, S D Bashkirov, Javier Rico, Ike Toriyama, Naoyuki Miyada, Hideki Yanagisawa, and Kensaku Ishizuka. 2021. Learning Bipedal Robot Locomotion from Human Movement. *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), 2797–2803. https://api.semanticscholar.org/CorpusID:235196006

Chen Tessler, Yunrong Guo, Ofir Nabati, Gal Chechik, and Xue Bin Peng. 2024. MaskedMimic: Unified Physics-Based Character Control Through Masked Motion Inpainting. *ACM Transactions on Graphics (TOG)* (2024).

Michiel van de Panne, Ryan Kim, and Eugene Flume. 1994. Virtual Wind-up Toys for Animation. In *Proceedings of Graphics Interface '94*. 208–215.

Kevin Wampler, Zoran Popović, and Jovan Popović. 2014. Generalizing Locomotion Style to New Animals with Inverse Optimal Regression. *ACM Trans. Graph.* 33, 4, Article 49 (July 2014), 11 pages.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2009. Optimizing Walking Controllers. In *ACM SIGGRAPH Asia 2009 Papers* (Yokohama, Japan) *(SIGGRAPH Asia '09)*. Association for Computing Machinery, New York, NY, USA, Article 168, 8 pages. https://doi.org/10.1145/1661412.1618514

Tingwu Wang, Yunrong Guo, Maria Shugrina, and Sanja Fidler. 2020. UniCon: Universal Neural Controller For Physics-based Character Motion. arXiv:2011.15119 [cs.GR]

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Trans. Graph.* 39, 4, Article 33 (jul 2020), 12 pages. https://doi.org/10.1145/3386569.3392381

Wayne Lewis Wooten. 1998. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. Ph. D. Dissertation. USA. Advisor(s) Hodgins, Jessica K. AAI9827367.

Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. 2020. Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 10607–10616. https://proceedings.mlr.press/v119/xu20h.html

Pei Xu and Ioannis Karamouzas. 2021. A GAN-Like Approach for Physics-Based Imitation Learning and Interactive Character Control. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 3, Article 44 (sep 2021), 22 pages. https://doi.org/10.1145/3480148

Pei Xu, Xiumin Shang, Victor Zordan, and Ioannis Karamouzas. 2023. Composite Motion Learning with Task Control. *ACM Trans. Graph.* 42, 4, Article 93 (July 2023), 16 pages. https://doi.org/10.1145/3592447

Heyuan Yao, Zhenhua Song, Yuyang Zhou, Tenglong Ao, Baoquan Chen, and Libin Liu. 2024. MoConVQ: Unified Physics-Based Motion Control via Scalable Discrete Representations. *ACM Trans. Graph.* 43, 4, Article 144 (July 2024), 21 pages. https://doi.org/10.1145/3658137

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

Ye Yuan, Shih-En Wei, Tomas Simon, Kris Kitani, and Jason M. Saragih. 2021. SimPoE: Simulated Character Control for 3D Human Pose Estimation. *2021 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR) (2021).

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. 2008. Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3* (Chicago, Illinois) *(AAAI'08)*. AAAI Press, 1433–1438.

Victor Brian Zordan and Jessica K. Hodgins. 2002. Motion Capture-Driven Simulations That Hit and React. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Antonio, Texas) *(SCA '02)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/545261.545276

Supplementary Material

Table of Contents

A Didactic Example

We include a didactic example to analyze the behavior of ADD. In this experiment, we apply ADD to a simple regression task, where the goal is to approximate a 1D target function $f(x) = \cos(x^{2.5})$, visualized in Figure 10. The target function is designed to present a function approximator with varying levels of difficulty in different regions of the input space. Near the origin, where the oscillation frequency is low, $f(x) = \cos(x^{2.5})$ is easy to model. However, as the frequency increases farther from the origin, the function becomes progressively difficult to model. This variation in the smoothness of the target function presents a testbed for examining the adaptive behavior of ADD to attend to different sub-objectives based on their relative difficulty.

A.1 Experimental Setup

We uniformly sample $N = 512$ points from $f(x)$ within the interval $[0, 4.3]$ to build the training dataset. A network $G(x)$ is trained to predict an output y_i that approximates $\hat{y} = f(x)$ for every x_i in the dataset where $1 \leq i \leq N$. The prediction errors for the entire dataset are then aggregated into a vector,

$$\Delta = \begin{bmatrix} \hat{y}_1 - y_1 \\ \vdots \\ \hat{y}_N - y_N \end{bmatrix},$$

and provided as input to the discriminator $D(\Delta)$. Both $G(x)$ and $D(\Delta)$ consist of 2 hidden layers with 1024 and 512 units, respectively. Table 4 provides a detailed documentation of the hyperparameters used in this experiment.

A.2 Results

Figure 10 demonstrates that the regression model trained through ADD provides a reasonable approximation of the ground truth target function. To analyze the discriminator's behavior in dynamically weighting objectives over the course of training, Figure 11 depicts the gradients of the discriminator's output with respect to the prediction error for each sample $\nabla_\Delta D(\Delta)$ at different training iterations. The N prediction errors are the objectives of this MOO problem, and the gradients $\nabla_\Delta D(\Delta)$ can be interpreted as the weights assigned by $D(\Delta)$ to each objective. At the beginning of training, the discriminator assigns random weights to the objectives. As training progresses, and $G(x)$ learns how to fit $f(x)$ accurately near the origin, the discriminator automatically begins to assign higher weights to regions farther from the origin, which exhibit larger prediction errors. ADD's adaptive weighting of different objectives ensures that more difficult objectives receive more focus as training progresses, preventing the easier objectives from dominating the overall loss.

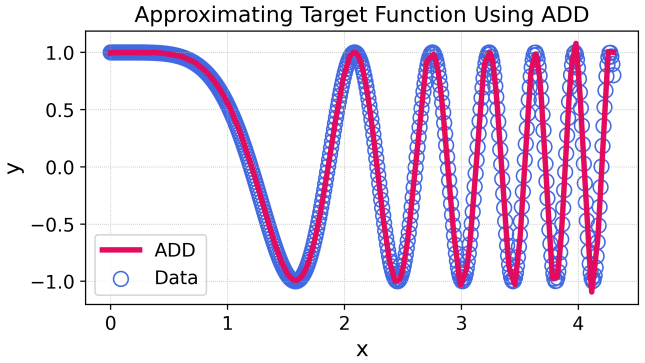


Fig. 10. Visualization of predictions made by ADD (red line) using data points (blue circles) sampled from the target function $f(x) = \cos(x^{2.5})$. ADD approximates the ground truth data adequately.

Table 4. ADD regression experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	0.1
Generator Batch Size	512
D Discriminator Stepsize	10^{-5}
G Generator Stepsize	10^{-4}

B Motion Tracking Additional Details

B.1 Experimental Setup and Computational Cost

All experiments are conducted on Nvidia A100 GPUs, with physics simulations implemented using Isaac Gym [Makoviychuk et al. 2021]. Humanoid character simulations run at 120 Hz, and the policy is queried at 30 Hz. For most motion imitation tasks on the humanoid character (excluding DanceDB and LaFAN1), policies are trained with approximately 750–850 million samples, requiring about 9 h for ADD, 12 h for AMP, and 6.5 h for DeepMimic. On the larger DanceDB dataset [Mahmood et al. 2019], all methods are trained with roughly 9 billion samples, taking 5 days for ADD, 6 days for AMP, and 3.5 days for DeepMimic. For the LaFAN1 subset [Harvey et al. 2020], training is extended to 18 billion samples, requiring 10 days for ADD, 7 days for DeepMimic, and 11 days for AMP.

Each EVAL robot policy is trained with approximately 900 million samples, requiring a wall-clock time of around 16 h for ADD, 18 h for

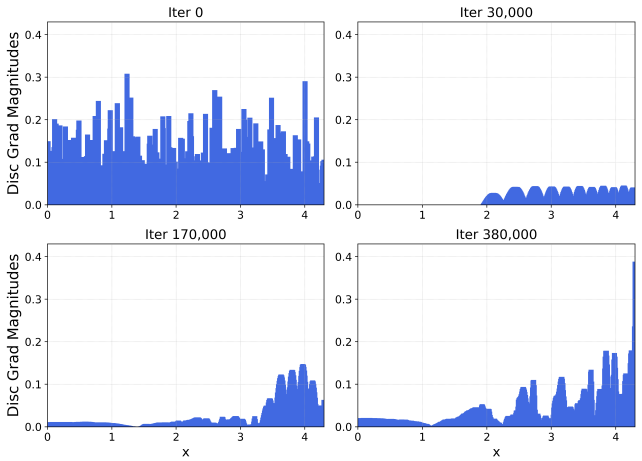


Fig. 11. Magnitudes of the gradients of the discriminator's output with respect to the prediction error for each sample at different training iterations. As training progresses and $G(x)$ gradually learns to approximate samples near the origin, the discriminator assigns higher and higher weights to samples farther from the origin, essentially honing in on the more difficult objectives.

AMP, and 13 h for DeepMimic. The robot simulations are performed at a simulation frequency of 150 Hz, with the control frequency set to 50 Hz.

B.2 DeepMimic's Imitation Reward Function

Peng et al. [2018] employ a manually-designed motion imitation reward function composed of five sub-terms,

$$r_t^{\text{DM}} = w^p r_t^p + w^{jv} r_t^{jv} + w^{rv} r_t^{rv} + w^e r_t^e + w^c r_t^c. \tag{14}$$

The pose reward r_t^p computes the difference between the joint orientation quaternions of the simulated character $\mathbf{q}_t^j$ and those of the reference motion $\hat{\mathbf{q}}_t^j$,

$$r_t^p = \exp\left[-\alpha^p \left(\sum_j \left\|\hat{\mathbf{q}}_t^j \ominus \mathbf{q}_t^j\right\|^2\right)\right]. \tag{15}$$

Here, α^p is a manually-tuned hyperparameter. The joint velocity reward r_t^{jv} is computed from the difference of local joint velocities, with $\dot{\mathbf{q}}_t^j$ being the angular velocity of the j-th joint,

$$r_t^{jv} = \exp\left[-\alpha^{jv} \left(\sum_j \left\|\hat{\dot{\mathbf{q}}}_t^j - \dot{\mathbf{q}}_t^j\right\|^2\right)\right]. \tag{16}$$

The root velocity reward r_t^{rv} measures the difference between the root velocities of the simulated character and that of the reference motion, with $\mathbf{v}_t^r$ representing the linear and angular root velocity,

$$r_t^{rv} = \exp\left[-\alpha^{rv} \left(\left\|\hat{\mathbf{v}}_t^r - \mathbf{v}_t^r\right\|^2\right)\right]. \tag{17}$$

The end-effector reward r_t^e calculates the difference in the end-effector positions of the simulated character $\mathbf{p}_t^e$ and those of the reference motion $\hat{\mathbf{p}}_t^e$,

$$r_t^e = \exp\left[-\alpha^e \left(\sum_e \left\|\hat{\mathbf{p}}_t^e - \mathbf{p}_t^e\right\|^2\right)\right]. \tag{18}$$

Finally, r_t^c encourages the character's center-of-mass $\mathbf{p}_t^c$ to match the center-of-mass of the reference motion $\hat{\mathbf{p}}_t^c$,

$$r_t^c = \exp\left[-\alpha^c \left(\left\|\hat{\mathbf{p}}_t^c - \mathbf{p}_t^c\right\|^2\right)\right]. \tag{19}$$

In the implementation, every joint orientation or velocity error is also multiplied by a joint-specific weight in the calculation of r_t^p and r_t^{jv}.

Table 5. DeepMimic sensitivity analysis parameter settings. Reward weights are listed in the order of w^p, w^{jv}, w^{rv}, w^e, w^c, and reward scales are listed in the order of α^p, α^{jv}, α^{rv}, α^e, α^c. * denotes the final parameter setting used in the experiments.

Setting	Parameters	
	Reward Weights	Reward Scales
Setting 1	0.2, 0.2, 0.2, 0.2, 0.2	1, 1, 1, 1, 1
Setting 2	0.5, 0.1, 0.15, 0.1, 0.15	4, 10, 0.2, 1, 0.1
Setting 3	0.5, 0.1, 0.15, 0.1, 0.15	0.2, 0.05, 3, 1.5, 8
Setting 4	0.5, 0.1, 0.15, 0.1, 0.15	10, 0.04, 100, 7.5, 75
Setting 5	0.2, 0.1, 0.2, 0.05, 0.45	0.25, 0.01, 5, 1, 10
Default*	0.5, 0.1, 0.15, 0.1, 0.15	0.25, 0.01, 5, 1, 10

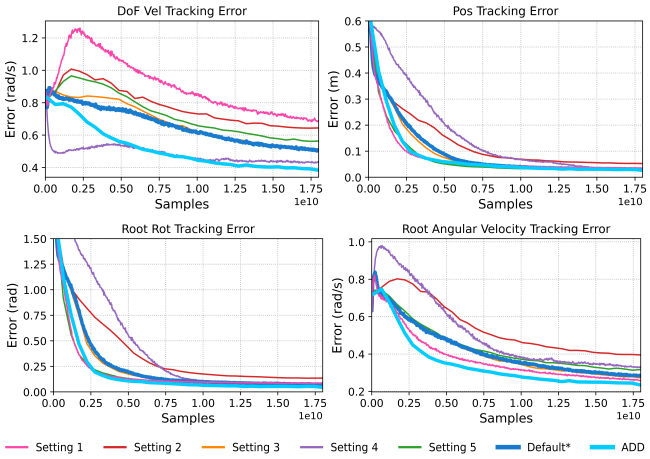


Fig. 12. Learning curves comparing the tracking performance of DeepMimic, under different parameter settings, with ADD on a subset of the LaFAN1 dataset [Harvey et al. 2020]. Table 5 provides more details on the parameter settings. DeepMimic's performance is sensitive to its hyperparameters. Some configurations can lead to poor tracking performance and sample efficiency. Some settings lead to improvement on certain metrics but degrade others. Selecting an optimal set of hyperparameters requires domain knowledge and significant manual tuning. In contrast, ADD automatically balances competing objectives during training, consistently achieving superior tracking accuracy and sample efficiency across metrics.

Table 6. ADD humanoid character motion imitation experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	0.1 (Climb)
	1 (Other)
K Update Minibatch Size	4096×4
π Policy Stepsize	1×10^{-4}
V Value Stepsize	1×10^{-4}
D Discriminator Stepsize	5×10^{-4}
$\mathcal{B}$ Experience Buffer Size	4096×32
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2

Table 7. DeepMimic humanoid character motion imitation experiment hyperparameters.

Parameter	Value
K Update Minibatch Size	4096×4
π Policy Stepsize	5×10^{-5}
V Value Stepsize	5×10^{-5}
$\mathcal{B}$ Experience Buffer Size	4096×32
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2
w^p, w^{jv}, w^{rv}, w^e, w^c Reward Weights	0.5, 0.1, 0.15, 0.1, 0.15
α^p, α^{jv}, α^{rv}, α^e, α^c Reward Scales	0.25, 0.01, 5.0, 1.0, 10.0
Joint Weights	1.0, 0.6, 0.6, 0.4, 0.0, 0.6, 0.4, 0.0, 1.0, 0.6, 0.4, 1.0, 0.6, 0.4

Table 8. AMP humanoid character motion imitation experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	5.0
K Update Minibatch Size	4096×4
π Policy Stepsize	5×10^{-5}
V Value Stepsize	5×10^{-5}
D Discriminator Stepsize	2.5×10^{-4}
$\mathcal{B}$ Experience Buffer Size	4096×32
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2

B.3 DeepMimic Sensitivity Analysis

To evaluate how sensitive DeepMimic is to its reward hyperparameters, we use DeepMimic to train simulated humanoid characters to imitate motions from the LaFAN1 subset using six different parameter settings. Details of the six settings are provided in Table 5, and the resulting learning curves are shown in Figure 12.

The results show considerable variation in both tracking performance and sample efficiency across settings. As shown in Figure 12, Setting 4 achieves strong tracking accuracy, but demonstrates poor sample efficiency. Setting 1 produces motions with accurate root rotation, root angular velocity, and position tracking, but suffers from high DoF velocity tracking errors, which lead to visibly jittery motions. Setting 2 performs poorly across all metrics and learns slowly, suggesting that certain parameter choices can significantly degrade both sample efficiency and tracking quality. In contrast, Settings 3, 5, and default*—sharing similar parameter configurations—consistently achieve great tracking accuracy and learning efficiency. Based on this observation, Setting default* is selected for the humanoid motion imitation experiments. Meanwhile, ADD achieves superior tracking accuracy and sample efficiency across all metrics without requiring manual tuning.

These findings highlight a key limitation of DeepMimic: its performance is highly sensitive to the choice of hyperparameters. While some configurations may yield strong results, finding them often requires extensive domain knowledge and iterative tuning. Poorly chosen parameters can result in degraded motion quality or inefficient learning, which presents a challenge for generalizing DeepMimic to new tasks or embodiments. In contrast, ADD's ability to automatically balance objectives leads to more robust and efficient learning across metrics, reducing the burden of manual design.

B.4 Hyperparameter Settings

Tables 6, 7, 8 list the hyperparameters of ADD, DeepMimic, and AMP used in the humanoid character motion imitation experiments. Similarly, Tables 9, 10, 11 provide the hyperparameters used in the EVAL robot motion imitation experiments.

DeepMimic has significantly more hyperparameters to be tuned to achieve good tracking performance than ADD and AMP. Moreover, these weights need to be re-tuned when switching to characters of different morphologies, as shown in Tables 7 and 10. In contrast, many fewer hyperparameters need to be adjusted for ADD when switching to different characters.

B.5 Learning Curves

Figures 13 and 14 present the learning curves of DeepMimic, AMP, and ADD across all skills for the EVAl robot and the humanoid character, respectively.

B.6 Steering Task Reward Functions

For ADD, task objectives are appended to the differential vector Δ_t,

$$\Delta_t = \begin{bmatrix} \Delta_t^{\text{tracking}} \\ v^* - \mathbf{v}_t^T \mathbf{d}_t^* \\ -\left\| \mathbf{v}_t - (\mathbf{v}_t^T \mathbf{d}_t^*) \mathbf{d}_t^* \right\| \end{bmatrix},$$

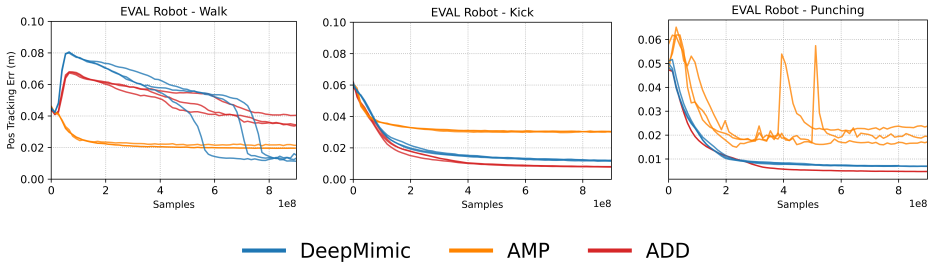


Fig. 13. Learning curves illustrating the position tracking errors of simulated robots trained using AMP [Peng et al. 2021], DeepMimic [Peng et al. 2018], and our method ADD. Statistics over 3 training runs, initialized with different random seeds, are depicted. Without relying on manually designed reward functions, ADD enables simulated robots to reproduce a set of challenging skills with quality on par with DeepMimic, a state-of-the-art motion-tracking method.

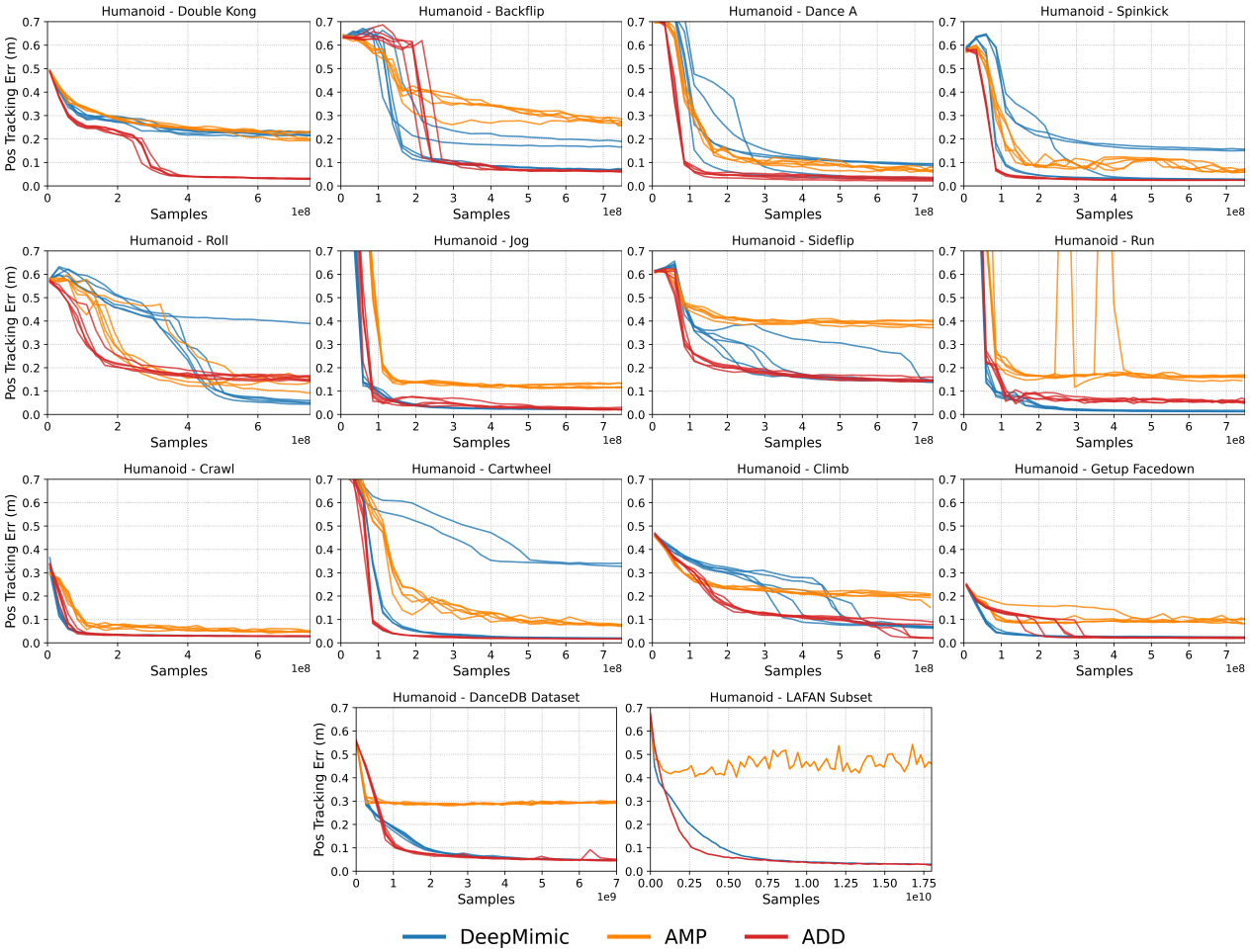


Fig. 14. Learning curves comparing the tracking performance of simulated humanoid characters trained via AMP [Peng et al. 2021], DeepMimic [Peng et al. 2018], and our method ADD. Statistics are computed over 5 training runs initialized with different random seeds, except for the LaFAN1 subset (1 run due to computational cost). ADD is capable of learning highly agile and acrobatic skills, achieving comparable tracking performance and sample efficiency to DeepMimic, without requiring manual reward engineering. Moreover, ADD exhibits better consistency across seeds, whereas DeepMimic often converges to suboptimal behaviors in some seeds.

Table 9. ADD EVAL robot motion imitation experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	1
$\mathcal{B}$ Experience Buffer Size	4096×32
K Update Minibatch Size	4096×4
π Policy Stepsize	8×10^{-5}
V Value Stepsize	8×10^{-5}
D Discriminator Stepsize	4×10^{-4}
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2

Table 10. DeepMimic EVAL robot motion imitation experiment hyperparameters.

Parameter	Value
K Update Minibatch Size	4096×4
π Policy Stepsize	1×10^{-4}
V Value Stepsize	1×10^{-4}
$\mathcal{B}$ Experience Buffer Size	4096×32
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2
$w^p, w^{jv}, w^{rv}, w^e, w^c$ Reward Weights	0.5, 0.05, 0.5, 0.15, 0.1
$\alpha^p, \alpha^{jv}, \alpha^{rv}, \alpha^e, \alpha^c$ Reward Scales	1.0, 0.01, 10.0, 1.0, 10.0
Joint Weights	1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0

Table 11. AMP EVAL robot motion imitation experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	0.1
K Update Minibatch Size	4096×4
π Policy Stepsize	8×10^{-5}
V Value Stepsize	8×10^{-5}
D Discriminator Stepsize	8×10^{-5}
$\mathcal{B}$ Experience Buffer Size	4096×32
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2

and are amplified by a factor 50 after normalization. The reward is then simply $r_t = -\log(1 - D(\Delta_t))$. For AMP and DeepMimic, the reward is changed to

$$r_t = 0.5\, r_t^{\text{tracking}} + 0.5\, r_t^G, \tag{20}$$

where

$$r_t^G = \exp\left[-2\left((v^* - \mathbf{v}_t^T \mathbf{d}_t^*)^2 + 0.1 \left\|\mathbf{v}_t - (\mathbf{v}_t^T \mathbf{d}_t^*)\mathbf{d}_t^*\right\|^2\right)\right]. \tag{21}$$

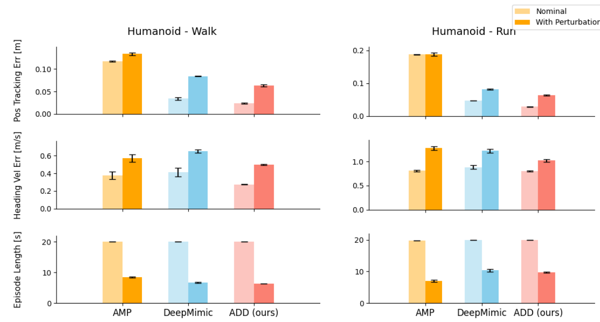


Fig. 15. Robustness of locomotion policies in Sec. 6.5 under random external force perturbations. Policies are trained without perturbations and stress-tested by randomly applying forces up to 300 N during inference. Performance is reported in terms of the episode length until loss of balance, as well as position tracking error and target velocity error before falling. Lighter bars denote nominal performance, and darker bars indicate performance under perturbation. ADD exhibits levels of degradation comparable to DeepMimic, reflecting a similar degree of robustness.

B.7 Robustness to Perturbations

To assess the robustness of locomotion policies trained using our approach, we evaluate their performance under random force perturbations and compare the results with prior methods. We take the policies trained in Section 6.5 and randomly apply random forces of up to 300 N on the character's root. These perturbations are introduced only at test time; the policies are not exposed to such disturbances during training. Performance is measured with two criteria: (1) the duration the policy maintains balance before falling, and (2) the accuracy with which the policy follows the reference motion and steering commands while subjected to perturbations, prior to falling. As shown in Figure 15, ADD exhibits a performance degradation comparable to DeepMimic, indicating a similar degree of robustness to perturbations.

C Walker Additional Details

All policies are trained for approximately 60 million samples, which takes a wall-clock time of around 24 h for the manual rewards and 26 h for ADD on 2 CPU cores.

C.1 Reward Function

ADD's reward is $r_t = -\log(1 - D(\Delta_t))$, where Δ_t is simply:

$$\Delta_t = \begin{bmatrix} 1.2 - h_t \\ 1 - u_t \\ 8.0 - v_t \end{bmatrix},$$

with h_t being the walker's height, u_t the cosine of the torso angle, and v_t the walker's horizontal speed. 1.2, 1, and 8.0 are target values

Table 12. ADD walker experiment hyperparameters.

Parameter	Value
λ^{GP} Gradient Penalty	0.001
K Update Minibatch Size	512
π Policy Stepsize	5×10^{-4}
V Value Stepsize	5×10^{-4}
D Discriminator Stepsize	5×10^{-5}
$\mathcal{B}$ Experience Buffer Size	4096
γ Discount	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.02

from the problem definition, not tunable hyperparameters. Compared to the reward function in Tassa et al. [2018], ADD employs a much simpler formulation with significantly fewer manually-tuned hyperparameters.

The manually-designed reward function used in Tassa et al. [2018] is as follows:

$$\begin{aligned} r_{\text{manual}} &= r_{\text{stand}} \times \frac{5 \cdot r_{\text{move}} + 1}{6} \\ r_{\text{stand}} &= \frac{3 \cdot \text{tol}\left(h_t; 1.2, \infty, 0.1, 0.6, \text{gaussian}\right) + \frac{1+u_t}{2}}{4} \\ r_{\text{move}} &= \text{tol}\left(v_t; 8.0, \infty, 0.5, 4.0, \text{linear}\right) \end{aligned} \tag{22}$$

The tolerance function is defined as:

$$\text{tol}(x; a, b, v_m, m, \text{sigmoid}) = \begin{cases} 1, & \text{if } a \le x \le b \\ f(d(x; a, b, m); v_m, \text{sigmoid}), & \text{otherwise} \end{cases}, \tag{23}$$

where

$$d(x; a, b, m) = \frac{\max(a - x, x - b)}{m}.$$

Here, $[a, b]$ represents the bounds, m is the margin, v_m is the value at margin, and sigmoid is the type of sigmoid function.

If sigmoid = linear:

$$f(d; v_m, \text{linear}) = \begin{cases} 1 - (1 - v_m) \cdot d, & \text{if } |(1 - v_m) \cdot d| < 1 \\ 0, & \text{otherwise} \end{cases}. \tag{24}$$

If sigmoid = gaussian:

$$f(d; v_m, \text{gaussian}) = \exp\left(-\ln\left(\frac{1}{v_m}\right) \cdot d^2\right). \tag{25}$$

C.2 Hyperparameters

The hyperparameters for ADD in the Walker task are documented in Table 12.

D Go1 Additional Details

All policies are trained for approximately 1 billion samples, which takes about 3 hours for both ADD and the manually designed reward baseline on an A100 GPU. During training, early termination is applied in both methods when the torso, thigh, or hip of the quadruped makes contact with the ground.

D.1 Reward Function

The reward for ADD is given by $r_t = -\log(1 - D(\Delta_t))$, where Δ_t is:

$$\Delta_t = \begin{bmatrix} 0 - \mathbf{v}_{b,z} \\ 0 - \|\boldsymbol{\omega}_{b,xy}\|^2 \\ 0 - \|\mathbf{z}_{xy}\|^2 \\ 0.3 - h_b \\ 500 - \max\left(\|\boldsymbol{\tau}_j\|^2, 500\right) \\ 100 - \max\left(\|\dot{\mathbf{q}}_j\|^2, 100\right) \\ 100000 - \max\left(\|\ddot{\mathbf{q}}_j\|^2, 100000\right) \\ 0.5 - \max\left(\|\dot{\mathbf{q}}_j^*\|^2, 0.5\right) \\ \mathbf{v}_{b,x}^* - \mathbf{v}_{b,x} \\ \mathbf{v}_{b,y}^* - \mathbf{v}_{b,y} \\ \boldsymbol{\omega}_{b,z}^* - \boldsymbol{\omega}_{b,z} \\ 0.02 - \sum_{f=0}^{4} t_{\text{air},f} \end{bmatrix}. \tag{26}$$

The notations follow the definitions in Table 13. The z axis is aligned with gravity.

Table 13. Definition of symbols.

Joint positions	$\mathbf{q}_j$
Joint velocities	$\dot{\mathbf{q}}_j$
Joint accelerations	$\ddot{\mathbf{q}}_j$
Target joint positions	$\mathbf{q}_j^*$
Joint torques	$\boldsymbol{\tau}_j$
Base linear velocity	$\mathbf{v}_b$
Base angular velocity	$\boldsymbol{\omega}_b$
Base height	h_b
Commanded base linear velocity	$\mathbf{v}_b^*$
Commanded base angular velocity	$\boldsymbol{\omega}_b^*$
Number of collisions	n_c
Feet air time	$\mathbf{t}_{\text{air}}$
Environment time step	dt
Gravity vector	$\mathbf{z}$

The manual reward function we compare to is provided in Table 14. The reward terms are taken from what Rudin et al. [2022] used in their open-source repository. The weights are largely similar to the weights they used for the A1 robot, with some weights tuned to achieve better performance on the Go1 robot. Extensive hyperparameter tuning was performed to ensure the baseline method achieves good performance. ADD has fewer tunable hyperparameters than the manually-designed reward function.

D.2 Hyperparameters

Training hyperparameters of ADD and the manual reward function are detailed in Table 15 and 16, respectively.

Table 14. Definition of reward terms used in [Rudin et al. 2022], with $\phi(x) := \exp\left(-\frac{\|x\|^2}{0.25}\right)$. The z axis is aligned with gravity.

	definition	**weight**
Linear velocity tracking	$\phi(\mathbf{v}^*_{b,xy} - \mathbf{v}_{b,xy})$	$1\,dt$
Angular velocity tracking	$\phi(\boldsymbol{\omega}^*_{b,z} - \boldsymbol{\omega}_{b,z})$	$0.5\,dt$
Linear velocity penalty	$-\mathbf{v}^2_{b,z}$	$2\,dt$
Angular velocity penalty	$-\|\boldsymbol{\omega}_{b,xy}\|^2$	$0.05\,dt$
Orientation penalty	$-\|\mathbf{z}_{xy}\|^2$	$0\,dt$
Root Height	$-(0.3 - h_b)^2$	$50\,dt$
Joint velocity	$-\|\dot{\mathbf{q}}_j\|^2$	$0\,dt$
Joint acceleration	$-\|\ddot{\mathbf{q}}_j\|^2$	$2.5 \times 10^{-7}\,dt$
Joint torques	$-\|\boldsymbol{\tau}_j\|^2$	$0.0002\,dt$
Action rate	$-\|\dot{\mathbf{q}}^*_j\|^2$	$0.01\,dt$
Feet air time	$\sum_{f=0}^{4}(\mathbf{t}_{\text{air},f} - 0.2)$	$1\,dt$

Table 15. ADD Go1 quadruped training hyperparameters.

Parameter	**Value**
λ^{GP} Gradient Penalty	0.1
K Update Minibatch Size	4096×6
π Policy Stepsize	1×10^{-4}
V Value Stepsize	1×10^{-4}
D Discriminator Stepsize	5×10^{-4}
$\mathcal{B}$ Experience Buffer Size	4096×24
γ Discount	0.99
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.02

Table 16. Manual reward Go1 quadruped training hyperparameters.

Parameter	**Value**
K Update Minibatch Size	4096×6
π Policy Stepsize	1×10^{-4}
V Value Stepsize	1×10^{-4}
$\mathcal{B}$ Experience Buffer Size	4096×24
γ Discount	0.99
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.02

D.3 Full Learning Curves

The comprehensive collection of learning curves on the quadrupedal task can be found in Figure 16.

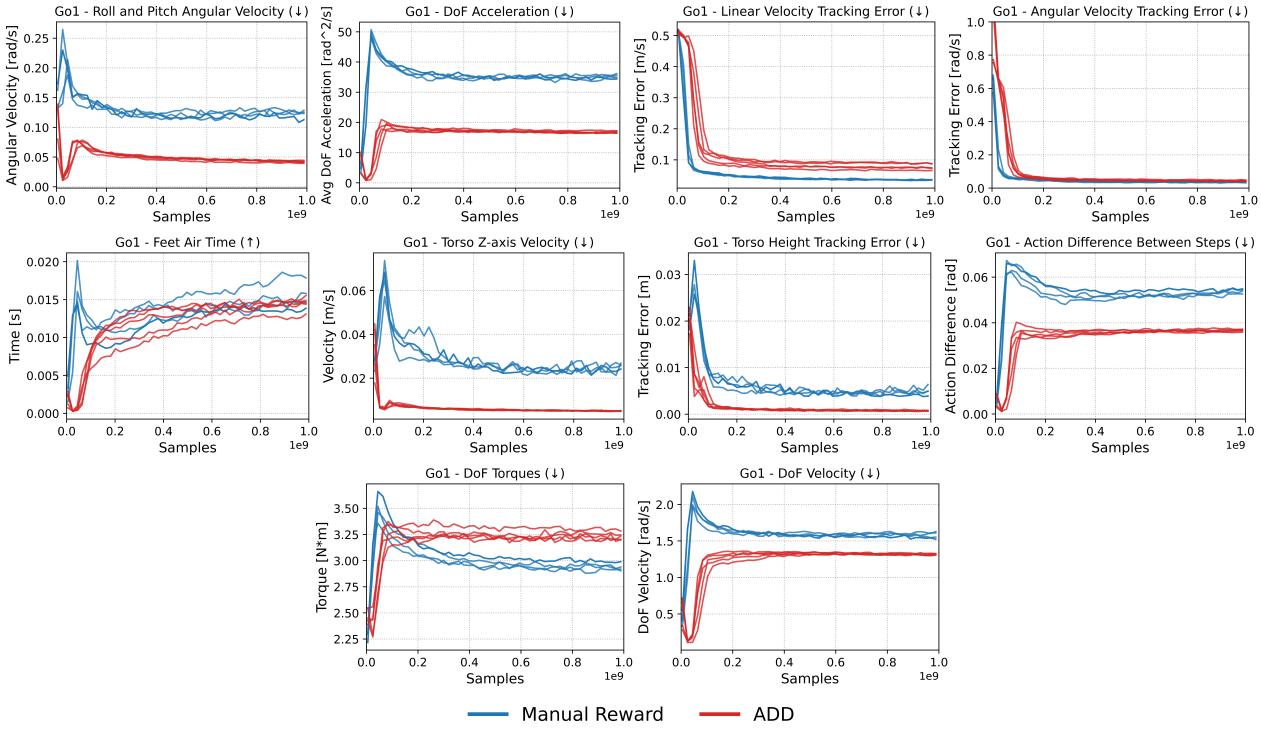


Fig. 16. Full learning curves for all training objectives on the quadruped task, with results averaged over five training runs initialized with different random seeds. ADD outperforms the manually designed reward function from Rudin et al. [2022] on several metrics associated with torso stability and smooth control. Overall, ADD achieves comparable sample efficiency, final performance, and consistency to the manually-designed reward function.