AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control

XUE BIN PENG†, University of California, Berkeley
ZE MA†, Shanghai jiao Tong University
PIETER ABBEEL, University of California, Berkeley
SERGEY LEVINE, University of California, Berkeley
ANGJOO KANAZAWA, University of California, Berkeley

1 INTRODUCTION

Synthesizing natural and life-like motions for virtual characters is a crucial element for breathing life into immersive experiences, such as films and games. The demand for realistic motions becomes even more apparent for VR applications, where users are provided with rich modalities through which to interact with virtual agents. Developing control strategies that are able to replicate the properties of naturalistic behaviors is also of interest for robotic systems, as natural motions implicitly encode important properties, such as safety and energy efficiency, which are vital for effective operation of robots in the real world. While examples of natural motions are commonplace, identifying the underlying characteristics that constitute these behaviors is nonetheless challenging, and more difficult still to replicate in a controller.

So what are the characteristics that constitute natural and life-like behaviors? Devising quantitative metrics of the naturalness of motions has been a fundamental challenge for optimization-based character animation techniques [Al Borno et al. 2013; Wampler et al. 2014; Wang et al. 2009]. Heuristics such as symmetry, stability, and effort minimization can improve the realism of motions produced by physically simulated characters [Grochow et al. 2004; Mordatch et al. 2012, 2013; Yu et al. 2018]. But these strategies may not be broadly applicable to all behaviors of interest. Effective applications of these
heuristics often require careful balancing of the various objectives, a tuning process that may need to be repeated for each task. Data-driven methods are able to mitigate some of these challenges by leveraging motion clips recorded from real-life actors to guide the behaviors of simulated characters [Da Silva et al. 2008; Liu et al. 2010; Muco et al. 2009; Sok et al. 2007]. A common instantiation of this approach is to utilize a tracking objective that encourages a character to follow particular reference trajectories relevant for a given task. These tracking-based methods can produce high-quality motions for a large repertoire skills. But extending these techniques to effectively leverage large unstructured motion datasets remains challenging, since a suitable motion clip needs to be selected for the character to track at each time step. This selection process is typically performed by a motion planner, which generates reference trajectories for solving a particular task [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2017]. However, constructing an effective motion planner can itself be a challenging endeavour, and entails significant overhead to annotate and organize the motion clips for a desired task. For many applications, it is not imperative to exactly track a particular reference motion. Since a dataset typically provides only a limited collection of example motions, a character will inevitably need to deviate from the reference motions in order to effectively perform a given task. Therefore, the intent is often not for the character to closely track a particular motion, but to adopt general behavioral characteristics depicted in the dataset. We refer to these behavioral characteristics as a style.

In this work, we aim to develop a system where users can specify high-level task objectives for a character to perform, while the low-level style of a character’s movements can be controlled through examples provided in the form of unstructured motion clips. To control the style of a character’s motions, we propose adversarial motion priors (AMP), a method for imitating behaviors from raw motion clips without requiring any task-specific annotations or organization of the dataset. Given a set of reference motions that constitutes a desired motion style, the motion prior is modeled as an adversarial discriminator, trained to differentiate between behaviors depicted in the dataset from those produced by the character. The motion prior therefore acts as a general measure of similarity between the motions produced by a character and the motions in the dataset. By incorporating the motion prior in a goal-conditioned reinforcement learning framework, our system is able to train physically simulated characters to perform challenging tasks with natural and life-like behaviors. Composition of diverse behaviors emerges automatically from the motion prior, without the need for a motion planner or other mechanism for selecting which clip to imitate.

The central contribution of this work is an adversarial learning approach for physics-based character animation that combines goal-conditioned reinforcement with an adversarial motion prior, which enables the style of a character’s movements to be controlled via example motion clips, while the task is specified through a simple reward function. We present one of the first adversarial learning systems that is able to produce high-quality full-body motions for physically simulated characters. By combining the motion prior with additional task objectives, our system provides a convenient interface through which users can specify high-level directions for controlling a character’s behaviors. These task objectives allow our characters to acquire more complex skills than those demonstrated in the original motion clips. While our system is built on well-known adversarial imitation learning techniques, we propose a number of important design decisions that lead to substantially higher quality results than those achieved by prior work, enabling our characters to learn highly dynamic and diverse motors skills from unstructured motion data.

2 RELATED WORK

Developing systems that can synthesize natural motions for virtual characters is one of the fundamental challenges of computer animation. These procedural animation techniques can be broadly categorized as kinematic methods and physics-based methods. Kinematic methods generally do not explicitly utilize the equations of motion for motion synthesis. Instead, these methods often leverage datasets of motion clips to generate motions for a character [Lee et al. 2002, 2010b]. Given a motion dataset, controllers can be constructed to select an appropriate motion clip to play back for a particular scenario [Agrawal and van de Panne 2016; Safonova and Hodgins 2007; Treuille et al. 2007]. Data-driven methods using generative models, such as Gaussian processes [Levine et al. 2012; Ye and Liu 2010] and neural networks [Holden et al. 2017; Ling et al. 2020; Zhang et al. 2018], have also been applied to synthesize motions online. When provided with sufficiently large and high-quality datasets, kinematic methods are able to produce realistic motions for a large variety of sophisticated skills [Agrawal and van de Panne 2016; Lee et al. 2018, 2010b; Levine et al. 2011; Starke et al. 2019]. However, their ability to synthesize motions for novel situations can be limited by the availability of data. For complex tasks and environments, it can be difficult to collect a sufficient amount of data to cover all possible behaviors that a character may need to perform. This is particularly challenging for nonhuman and fictional creatures, where motion data can be scarce. In this work, we combine data-driven techniques with physics-based animation methods to develop characters that produce realistic and responsive behaviors to novel tasks and environments.

Physics-Based Methods: Physics-based methods address some of the limitations of kinematic methods by synthesizing motions from first principles. These methods typically leverage a physics simulation, or more general knowledge of the equations of motion, to generate motions for a character [Raibert and Hodgins 1991; Wampler et al. 2014]. Optimization techniques, such as trajectory optimization and reinforcement learning, play a pivotal role in many physics-based methods, where controllers that drive a character’s motions are produced by optimizing an objective function [Mordatch et al. 2012; Tan et al. 2014; van de Panne and Hodgins 1994]. While these methods are able to synthesize physically plausible motions for novel scenarios, even in the absence of motion data, designing effective objectives that lead to natural behaviors can be exceptionally difficult. Heuristics derived from prior knowledge of the characteristics of natural motions are commonly included into the objective function, such as symmetry, stability, effort minimization, and many more [Mordatch et al. 2012; Wang et al. 2009; Yu et al. 2018]. Simulating more biologically accurate actuators can also improve motion.

**Imitation Learning:** The challenges of designing objective functions that lead to natural motions have spurred the adoption of data-driven physics-based animation techniques [Da Silva et al. 2008; Kwon and Hodgins 2017; Lee et al. 2010a; Sharon and van de Panne 2005; Zordan and Hodgins 2002], which utilizes reference motion data to improve motion quality. Reference motions are typically incorporated through an imitation objective that encourages a character to imitate motions in the dataset. The imitation objective is commonly implemented as a tracking objective, which attempts to minimize the pose error between the simulated character and target poses from a reference motion [Lee et al. 2010a; Liu et al. 2016, 2010; Peng et al. 2018a; Sok et al. 2007]. Since the pose error is generally computed with respect to a single target pose at a time, some care is required to select an appropriate target pose from the dataset. A simple strategy is to synchronize the simulated character with a given reference motion using a phase variable [Lee et al. 2019; Peng et al. 2018a,b], which is provided as an additional input to the controller. The target pose at each time step can then be conveniently determined by selecting the target pose according to the phase. This strategy has been effective for imitating individual motion clips, but it can be difficult to scale to datasets containing multiple disparate motions, as it may not be possible to synchronize and align multiple reference motions according to a single-phase variable. Recent methods have extended these tracking-based techniques to larger motion datasets by explicitly providing target poses from the reference motion that is being tracked as inputs to the controller [Bergamin et al. 2019; Chentanez et al. 2018; Park et al. 2019; Won et al. 2020]. This then allows a controller to imitate different motions depending on the input target poses. However, selecting the appropriate motion for a character to imitate in a given scenario can still entail significant algorithmic overhead. These methods often require a high-level motion planner that selects which motion clip the character should imitate for a given task [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2017]. The character’s performance on a particular task can therefore be constrained by the performance of the motion planner.

Another major limitation of tracking-based imitation techniques is the need for a pose error metric when computing the tracking objective [Liu et al. 2010; Peng et al. 2018a; Sharon and van de Panne 2005]. These error metrics are often manually-designed, and it can be challenging to construct and tune a common metric that is effective across all skills that a character is to imitate. Adversarial imitation learning provides an appealing alternative [Abbeel and Ng 2004; Ho and Ermon 2016; Ziebart et al. 2008], where instead of using a manually-designed imitation objective, these algorithms train an adversarial discriminator to differentiate between behaviors generated by an agent from behaviors depicted in the demonstration data (e.g. reference motions). The discriminator then serves as the objective function for training a control policy to imitate the demonstrations. While these methods have shown promising results for motion imitation tasks [Merel et al. 2017; Wang et al. 2017], adversarial learning algorithms can be notoriously unstable and the resulting motion quality still falls well behind what has been achieved with state-of-the-art tracking-based techniques. Peng et al. [2019b] was able to produce substantially more realistic motions by regularizing the discriminator with an information bottleneck. However, their method still requires a phase variable to synchronize the policy and discriminator with the reference motion. Therefore, their results are limited to imitating a single motion per policy, and thus not suitable for learning from large diverse motion datasets. In this work, we propose an adversarial method for learning general motion priors from large unstructured datasets that contain diverse motion clips. Our approach does not necessitate any synchronization between the policy and reference motion. Furthermore, our approach does not require a motion planner, or any task-specific segmentation of the motion clips [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2017]. Instead, composition of multiple motions in furtherance of a task objective emerges automatically through the motion prior. We also present a number of design decisions for stabilizing the adversarial training process, leading to consistent and high-quality results.

**Latent Space Models:** Latent space models can also act as a form of motion prior that leads to more life-like behaviors. These models specify controls through a learned latent representation, which is then mapped to controls for the underlying system [Burgard et al. 2008; Florensa et al. 2017; Hausman et al. 2018; Heess et al. 2016]. The latent representation is typically learned through a pre-training phase using supervised learning or reinforcement learning techniques to encode a diverse range of behaviors into a latent representation. Once trained, this latent representation can be used to build a control hierarchy, where the latent space model acts as a low-level controller, and a separate high-level controller is trained to specify controls via the latent space [Florensa et al. 2017; Haarnoja et al. 2018; Lynch et al. 2020]. For motion control of simulated characters, the latent representation can be trained to encode behaviors from reference motion clips, which then constrains the behavior of a character to be similar to those observed in the motion data, therefore leading to more natural behaviors for downstream tasks [Merel et al. 2019; Peng et al. 2019a]. However, since the realism of the character’s motions is enforced implicitly through the latent representation, rather than explicitly through an objective function, it is still possible for the high-level controller to specify latent encodings that produce unnatural behaviors [Merel et al. 2020; Peng et al. 2019a]. Luo et al. [2020] proposed an adversarial domain confusion loss to prevent the high-level controller from specifying encodings that are different from those observed during pre-training. However, since this adversarial objective is applied in the latent space, rather than on the actual motions produced by the character, the model is nonetheless prone to generating unnatural behaviors. Our proposed motion prior directly enforces similarity between the motions produced by the character and those in the reference motion dataset, which enables our method to produce higher fidelity motions than what has been demonstrated by latent space models. Our motion prior also does not require a separate pre-training phase, and instead, can be trained jointly with the policy.
3 OVERVIEW

Given a dataset of reference motions and a task objective defined by a reward function, our system synthesizes a control policy that enables a character to achieve the task objective in a physically simulated environment, while utilizing behaviors that resemble the motions in the dataset. Crucially, the character’s behaviors need not exactly match any specific motion in the dataset, instead its movements need only to adopt more general characteristics exhibited by the corpus of reference motions. These reference motions collectively provide an example-based definition of a behavioral style, and by providing the system with different motion datasets, the character can then be trained to perform a task in a variety of distinct styles.

Figure 2 provides a schematic overview of the system. The motion dataset \( M \) consists of a collection of reference motions, where each motion \( m^i = \{ q^i_t \} \) is represented as a sequence of poses \( q^i_t \). The motion clips may be collected from the mocap of real-life actors or from artist-authored keyframe animations. Unlike previous frameworks, our system can be applied directly on raw motion data, without requiring task-specific annotations or segmentation of a clip into individual skills. The motion of the simulated character is controlled by a policy \( \pi(a_t|s_t, g) \) that maps the state of the character \( s_t \) and a given goal \( g \) to a distribution over actions \( a_t \). The actions from the policy specify target positions for proportional-derivative (PD) controllers positioned at each of the character’s joints, which in turn produce control forces that drive the motion of the character. The goal \( g \) specifies a task reward function \( r^G(s_t, a_t, s_{t+1}, g) \), which defines high-level objectives for the character to satisfy (e.g. walking in a target direction or punching a target). The style objective \( r^S(s_t, g) \) is specified by an adversarial discriminator, trained to differentiate between motions depicted in the dataset from motions produced by the character. The style objective therefore acts as a task-agnostic motion prior that provides an a-priori estimate of the naturalness or style of a given motion, independent of a specific task. The style objective then encourages the policy to produce motions that resemble behaviors depicted in the dataset.

4 BACKGROUND

Our system combines techniques from goal-conditioned reinforcement learning and generative adversarial imitation learning to train control policies that enable simulated characters to perform challenging tasks in a desired behavioral style. In this section, we provide a brief review of these techniques.

4.1 Goal-Conditioned Reinforcement Learning

Our characters are trained through a goal-conditioned reinforcement learning framework, where an agent interacts with an environment according to a policy \( \pi \) in order to fulfill a given goal \( g \in G \) sampled according to a goal distribution \( g \sim p(g) \). At each time step \( t \), the agent observes the state \( s_t \in S \) of the system, then samples an action \( a_t \in A \) from a policy \( a_t \sim \pi(a_t|s_t, g) \). The agent then applies that action, which results in a new state \( s_{t+1} \), as well as a scalar reward \( r_t = r(s_t, a_t, s_{t+1}, g) \). The agent’s objective is to learn a policy that maximizes its expected discounted return \( J(\pi) \),

\[
J(\pi) = \mathbb{E}_{\pi(a|s, g)} \left[ \sum_{t=0}^{T-1} \gamma^t r_t \right] ,
\]

where \( p(g) \) and \( \pi(a|s, g) \) represent the likelihood of a trajectory \( t = \{(s_t, a_t, r_t)\}_{t=0}^{T-1} \) under a policy \( \pi \) for a goal \( g \), \( p(s_0) \) is the initial state distribution, and \( p(s_{t+1}|s_t, a_t) \) represents the dynamics of the environment. \( T \) denotes the time horizon of a trajectory, and \( \gamma \in (0, 1) \) is a discount factor.

4.2 Generative Adversarial Imitation Learning

Generative adversarial imitation learning (GAIL) [Ho and Ermon 2016] adapts techniques developed for generative adversarial networks (GAN) [Goodfellow et al. 2014] to the domain of imitation learning. In the interest of brevity, we exclude the goal \( g \) from the notation, but the following discussion readily generalizes to goal-conditioned settings. Given a dataset of demonstrations \( M = \{(s_i, a_i)\} \), containing states \( s_i \) and actions \( a_i \) recorded from an unknown demonstration policy, the objective is to train a policy \( \pi(a|s) \) that imitates the behaviors of the demonstrator. Behavioral cloning can be used to directly fit a policy to map from states observed in \( M \) to their corresponding actions using supervised learning [Bojarski et al. 2016; Pomerleau 1988]. However, if only a small amount of demonstrations are available, then behavioral cloning techniques are prone to drift [Ross et al. 2011]. Furthermore, behavioral cloning is not directly applicable in settings where the demonstration actions are not observable (e.g. reference motion data).

GAIL addresses some of the limitations of behavioral cloning by learning an objective function that measures the similarity between the policy and the demonstrations, and then updating \( \pi \) via reinforcement learning to optimize the learned objective. The objective is modeled as a discriminator \( D(s, a) \), trained to predict whether a given state \( s \) and action \( a \) is sampled from the demonstrations \( M \) or generated by running the policy \( \pi \),

\[
\arg \min_D \mathbb{E}_{d^M(s,a)} \left[ \log (D(s,a)) \right] - \mathbb{E}_{d^\pi(s,a)} \left[ \log (1 - D(s,a)) \right] .
\]

\( d^M(s,a) \) and \( d^\pi(s,a) \) denote the state-action distribution of the dataset and policy respectively. The policy is then trained using the
RL objective detailed in Equation 1, with rewards specified by

\[ r_t = -\log (1 - D(s_t, a_t)) \]. (3)

This adversarial training procedure can be interpreted as training a policy to produce states and actions that appear to the discriminator as being indistinguishable from the demonstrations. It can be shown that this objective minimizes the Jensen-Shannon divergence between \( d^M(s, a) \) and \( d^M(s, a) \) [Ke et al. 2019; Nowozin et al. 2016].

5 ADVERSARIAL MOTION PRIOR

In this work, we consider reward functions that consist of two components specifying: 1) what task a character should perform, and 2) how the character should go about performing that task,

\[ r(s_t, a_t, s_{t+1}, g) = w^G \| \Phi(s_t, a_t, s_t, g) + w^S r^S(s_t, s_{t+1}) \] . (4)

The \( \text{what} \) is represented by a task-specific reward \( r^G(s_t, a_t, s_t, g) \), which defines high-level objectives that a character should satisfy (e.g. moving to a target location). The \( \text{how} \) is represented through a learned task-agnostic style-reward \( r^S(s_t, s_{t+1}) \), which specifies low-level details of the behaviors that the character should adopt when performing the task (e.g. walking vs. running to a target). The two reward terms are combined linearly with weights \( w^G \) and \( w^S \). The task-reward \( r^G \) can be relatively intuitive and simple to design. However, it can be exceptionally difficult to design a style-reward \( r^S \) that leads a character to learn naturalistic behaviors, or behaviors that conform to a particular style. Learning effective style objectives will therefore be the primary focus of this work.

We propose to model the style-reward with a learned discriminator, which we refer to as an adversarial motion prior (AMP), by analogy to the adversarial pose priors that were previously proposed for vision-based pose estimation tasks [Kanazawa et al. 2018]. Unlike standard tracking objectives, which measure pose similarity with respect to a specific reference motion, the motion prior returns a general score indicating the similarity of the character’s motion to the motions depicted in the dataset, without explicitly comparing to a particular motion clip. Given a motion dataset, the motion prior is trained using the GAIL framework to predict whether a state transition \((s_t, s_{t+1})\) is a real sample from the dataset or a fake sample produced by the character. The motion prior is independent of the task-specific goal \( g \), therefore a single motion prior can be applied to multiple tasks, and different motion priors can be applied to train policies that perform the same task but in different styles. By combining GAIL with additional task objectives, our approach decouples task specification from style specification, thereby enabling our characters to perform tasks that may not be depicted in the original demonstrations. However, adversarial RL techniques are known to be highly unstable. In the following sections, we discuss a number of design decisions to stabilize the training process and produce higher fidelity results.

5.1 Imitation from Observations

The original formulation of GAIL requires access to the demonstrator’s actions [Ho and Ermon 2016]. However, when the demonstrations are provided in the form of motion clips, the actions taken by the demonstrator are unknown, and only states are observed in the data. To extend GAIL to settings with state-only demonstrations, the discriminator can be trained on state transitions \( D(s, s') \) instead of state-action pairs \( D(s, a) \) [Torabi et al. 2018].

\[ \arg\min_D \mathbb{E}_{d^M(s, s')} \left[ \log (D(s, s')) \right] - \mathbb{E}_{d^P(s, s')} \left[ \log (1 - D(s, s')) \right] . \] (5)

\( d^M(s, s') \) and \( d^P(s, s') \) denote the likelihood of observing a state transition from \( s \) to \( s' \) in the dataset \( M \) and by following policy \( \pi \) respectively. Note that if the demonstrator is different from the agent (e.g. a human actor), the observed state transitions may not be physically consistent for the agent, and therefore impossible for the agent to perfectly reproduce. Despite this discrepancy, we show that the discriminator still provides an effective objective for imitating a wide range of behaviors.

5.2 Least-Squares Discriminator

The standard GAN objective detailed in Equation 5 typically uses a sigmoid cross-entropy loss function. However, this loss tends to lead to optimization challenges due to vanishing gradients as the sigmoid function saturates, which can hamper training of the policy [Arjovsky et al. 2017]. A myriad of techniques have been proposed to address these optimization challenges [Arjovsky et al. 2017; Berthelot et al. 2017; Gulrajani et al. 2017; Karras et al. 2017; Kodali et al. 2017; Mescheder et al. 2018; Radford et al. 2015; Salimans et al. 2016]. In this work, we adopt the loss function proposed for least-squares GAN (LSGAN) [Mao et al. 2017], which has demonstrated more stable training and higher quality results for image synthesis tasks. The following objective is used to train the discriminator,

\[ \arg\min_D \mathbb{E}_{d^M(s, s')} \left[ (D(s, s') - 1)^2 \right] + \mathbb{E}_{d^P(s, s')} \left[ (D(s, s') + 1)^2 \right] . \] (6)

The discriminator is trained by solving a least-squares regression problem to predict a score of 1 for samples from the dataset and −1 for samples recorded from the policy. The reward function for training the policy is then given by

\[ r(s_t, s_{t+1}) = \max \left[ 0, 1 - 0.25(D(s_t, s_{t+1}) - 1)^2 \right] . \] (7)

The additional offset, scaling, and clipping are applied to bound the reward between \([0, 1]\), as is common practice in previous RL frameworks [Peng et al. 2018a, 2016; Tassa et al. 2018]. Mao et al. [2017] showed that this least-squares objective minimizes the Pearson \( \chi^2 \) divergence between \( d^M(s, s') \) and \( d^P(s, s') \).

5.3 Discriminator Observations

Since the discriminator specifies rewards for training the policy, selecting an appropriate set of features for use by the discriminator when making its predictions is vital to provide the policy with effective feedback. As such, before a state transition is provided as input to the discriminator, we first apply an observation map \( \Phi(s_t) \) that extracts a set of features relevant for determining the characteristics of a given motion. The resulting features are then used as inputs to the discriminator \( D(\Phi(s), \Phi(s')) \). The set of features include:

- Linear velocity and angular velocity of the root, represented in the character’s local coordinate frame.
- Local rotation of each joint.
- Local velocity of each joint.
• 3D positions of the end-effectors (e.g., hands and feet), represented in the character’s local coordinate frame.

The root is designated to be the character’s pelvis. The character’s local coordinate frame is defined with the origin located at the root, the x-axis oriented along the root link’s facing direction, and the y-axis aligned with the global up vector. The 3D rotation of each spherical joint is encoded using two 3D vectors corresponding to the normal and tangent in the coordinate frame. This rotation encoding provides a smooth and unique representation of a given rotation. This set of observation features for the discriminator is selected to provide a compact representation of the motion across a single state transition. The observations also do not include any task-specific features, thus enabling the motion prior to be trained without requiring task-specific annotation of the reference motions, and allowing motion priors trained with the same dataset to be used for different tasks.

5.4 Gradient Penalty

The interplay between the discriminator and generator in a GAN often results in unstable training dynamics. One source of instability is due to function approximation errors in the discriminator, where the discriminator may assign nonzero gradients on the manifold of real data samples [Mescheder et al. 2018]. These nonzero gradients can cause the generator to overshoot and move off the data manifold, instead of converging to the manifold, leading to oscillations and instability during training. To mitigate this phenomenon, a gradient penalty can be applied to penalize nonzero gradients on samples from the dataset [Gulrajani et al. 2017; Kodali et al. 2017; Mescheder et al. 2018]. We incorporate this technique to improve training stability. The discriminator objective is then given by:

$$\arg \min_{\mathcal{D}} \mathbb{E}_{s \sim \mathcal{M}} \left[ \mathcal{D}(\Phi(s), \Phi(s')) - 1 \right]^2 + \mathbb{E}_{s \sim \mathcal{S}} \left[ \mathcal{D}(\Phi(s), \Phi(s')) + 1 \right]^2 + \frac{\lambda}{2} \mathbb{E}_{s \sim \mathcal{M}} \left[ \left\| \nabla_{\phi} \mathcal{D}(\phi) \right\|_{2} \right]^2,$$

where $\mathcal{M}$ is a manually-specified coefficient. Note, the gradient penalty is calculated with respect to the observation features $\Phi(s, \Phi(s'))$, not the full set of state features $(s, s')$. As we show in our experiments, the gradient penalty is crucial for stable training and effective performance.

6 MODEL REPRESENTATION

Given a high-level task objective and a dataset of reference motions, the agent is responsible for learning a control policy that fulfills the task objectives, while utilizing behaviors that resemble the motions depicted in the dataset. In this section, we detail the design of various components of the learning framework.

6.1 States and Actions

The state $s_t$ consists of a set of features that describes the configuration of the character’s body. The features are similar to those used by Peng et al. [2018a], which include the relative positions of each link with respect to the root, the rotation of each link as represented using the 6D normal-tangent encoding, along with the link’s linear and angular velocities. All features are recorded in the character’s local coordinate system. Unlike previous systems, which synchronize the policy with a particular reference motion by including additional phase information in the state, such as scalar phase variables [Lee et al. 2019; Peng et al. 2018a] or target poses [Bergamin et al. 2019; Chentanez et al. 2018; Won et al. 2020], our policies are not trained to explicitly imitate any specific motion from the dataset. Therefore, no such synchronization or phase information is necessary.

Each action $a_t$ specifies target positions for PD controllers positioned at each of the character’s joints. For spherical joints, each target is specified in the form of a 3D exponential map $q \in \mathbb{R}^3$ [Grassia 1998], where the rotation axis $v$ and rotation angle $\theta$ can be determined according to:

$$v = \frac{q}{\|q\|_2}, \quad \theta = \|q\|_2.$$

This representation provides a more compact parameterization than the 4D axis-angle or quaternion representations used in prior systems [Peng et al. 2018a; Won et al. 2020], while also avoiding gimbal lock from parameterizations such as euler angles. Target rotations for revolute joints are specified as 1D rotation angles $\theta$.

6.2 Network Architecture

Each policy $\pi$ is modeled by a neural network that maps a given state $s_t$ and goal $g$ to a Gaussian distribution over actions $\pi(a_t | s_t, g) \sim \mathcal{N}(\mu(s_t, g), \Sigma)$, with an input-dependent mean $\mu(s_t, g)$ and a fixed diagonal covariance matrix $\Sigma$. The mean is specified by a fully-connected network with two hidden layers, consisting of 1024 and 512 ReLU [Nair and Hinton 2010], followed by a linear output.
layer. The values of the covariance matrix \( \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots) \) are manually-specified and kept fixed over the course of training. The value function \( V(s_t, g) \) and discriminator \( D(s_t, s_{t+1}) \) are modeled by separate networks with a similar architecture as the policy.

6.3 Training

Our policies are trained using a combination of GAIL [Ho and Ermon 2016] and proximal-policy optimization (PPO) [Schulman et al. 2017].

Algorithm 1 provides an overview of the training process. At each time step \( t \), the agent receives a task-reward \( r^T_t = r^T(s_t, a_t, s_{t+1}, g) \) from the environment, it then queries the motion prior for a style-reward \( r^g_t = r^g(s_t, s_{t+1}) \), computed according to Equation 7. The two rewards are combined according to Equation 4 to yield the reward for the particular timestep. Following the approach proposed by Peng et al. [2018a], we incorporate reference state initialization and early termination. Reference state initialization is applied by initializing the character to states sampled randomly from all motion clips in the dataset. Early termination is triggered on most tasks when any part of the character’s body, with exception of the feet, makes contact with the ground. This termination criteria is disabled for more contact-rich tasks, such as rolling or getting up after a fall.

Once a batch of data has been collected with the policy, the recorded trajectories are used to update the policy and value function. The value function is updated with target values computed using TD(\( \lambda \)) [Sutton and Barto 1998]. The policy is updated using advantages computed using GAE(\( \lambda \)) [Schulman et al. 2015]. Each trajectory recorded from the policy is also stored in a replay buffer \( \mathcal{B} \), containing trajectories from past training iterations. The discriminator is updated according to Equation 8 using minibatches of transitions \((s, s')\) sampled from the reference motion data set \( \mathcal{M} \) and transitions from the replay buffer \( \mathcal{B} \). The replay buffer helps to stabilize training by preventing the discriminator from overfitting to the most recent batch of trajectories from the policy.

7 TASKS

To evaluate AMP’s effectiveness for controlling the style of a character’s motions, we apply our framework to train complex 3D simulated characters to perform various motion control tasks using different motion styles. The characters include a 34 DoF humanoid, a 59 DoF T-Rex, and a 64 DoF dog. A summary of each task is provided below. Please refer to Appendix A for a more in-depth description of each task and their respective reward functions.

**Target Heading:** In this task, the character’s objective is to move along a target heading direction \( \mathbf{d}^* \) at a target speed \( \mathbf{v}^* \). The goal for the policy is specified as \( g_t = (\tilde{d}^*_t, \tilde{v}^*_t) \), with \( \tilde{d}^*_t \) being the target direction in the character’s local coordinate frame. The target speed is selected randomly between \( \mathbf{v}^* \in [1, 5] \text{m/s} \). For slower moving styles, such as Zombie and Stealthy, the target speed is fixed at 1m/s.

**Target Location:** In this task, the character’s objective is to move to a target location \( \mathbf{x}^* \). The goal \( g_t = (\tilde{d}^*_t, \tilde{v}^*_t) \) records the target location in the character’s local coordinate frame.

**Dribbling:** To evaluate our system on more complex object manipulation tasks, we train policies for a dribbling task, where the character’s objective is to dribble a soccer ball to a target location. The goal \( g_t = (\tilde{x}^*_t, \tilde{h}^*_t) \) records the relative position of the target location with respect to the character. The state \( s_t \) is augmented with additional features that describe the state of the ball, including the position \( \tilde{z}^\text{ball}_t \), orientation \( \tilde{q}^\text{ball}_t \), linear velocity \( \tilde{v}^\text{ball}_t \), and angular velocity \( \tilde{\omega}^\text{ball}_t \) of the ball in the character’s local coordinate frame.

**Strike:** To demonstrate AMP’s ability to compose diverse behaviors, we consider a task where the character’s objective is to strike a target using a designated end-effector (e.g. hands). The target may be located at various distances from the character. Therefore, the character must first move close to the target before striking it. These distinct phases entail different optimal behaviors, and thus require the policy to compose and transition between the appropriate skills. The goal \( g_t = (\tilde{x}^*_t, \tilde{h}_t) \) records the location of the target \( \tilde{x}^*_t \) in the character’s local coordinate frame, along with an indicator variable \( h_t \) that specifies if the target has already been hit.

**Obstacles:** Finally, we consider tasks that involve visual perception and interaction with more complex environments, where the character’s objective is to traverse an obstacle-filled terrain, while maintaining a target speed. Policies are trained for two types of environments: 1) An environment containing a combination of obstacles include gaps, steps, and overhead obstructions that the character must duck under. 2) An environment containing narrow stepping stones that requires more precise contact planning. Examples of the environments are available in Figure 1 and 3. In order for the policy to perceive the upcoming obstacles, the state is augmented with a 1D height-field of the upcoming terrain.

8 RESULTS

We evaluate our framework’s effectiveness on a suite of challenging motion control tasks with complex simulated characters. First, we demonstrate that our approach can readily scale to large unstructured datasets containing diverse motion clips, which then enables our characters to perform challenging tasks in a natural and live-like manner by imitating behaviors from the dataset. The characters automatically learn to compose and generalize different skills from the motion data in order to fulfill high-level task objectives, without requiring mechanisms for explicit motion selection. We then evaluate AMP on a single-clip imitation task, and show that our method is able to closely imitate a diverse corpus of dynamic and acrobatic skills, producing motions that are nearly indistinguishable from reference motions recorded from human actors. Behaviors learned by the characters can be viewed in the supplementary video.

8.1 Experimental Setup

All environments are simulated using the Bullet physics engine [Coumans et al. 2013], with a simulation frequency of 1.2kHz. The policy is queried at 30Hz, and each action specifies target positions for PD controllers positioned at the character’s joints. All neural networks are implemented using Tensorflow [Abadi et al. 2015]. The gradient penalty coefficient is set to \( \psi_{\text{GP}} = 10 \). Detailed hyperparameter settings are available in Appendix B. Reference motion clips are collected from a combination of public mocap libraries, custom recorded mocap clips, and artist-authored keyframe animations.
Fig. 3. The motion prior can be trained with large datasets of diverse motions, enabling simulated characters to perform complex tasks by composing a wider range of skills. Each environment is denoted by “Character: Task (Dataset).”

[CMU [n.d.]; SFU [n.d.]; Zhang et al. 2018]. Depending on the task and character, each policy is trained with 100-300 million samples, requiring approximately 30-140 hours on 16 CPU cores. Code for our system will be released upon publication of this paper.

8.2 Tasks

In this section, we demonstrate AMP’s effectiveness for controlling the style of a character’s motions as it performs other high-level tasks. The weights for the task-reward and style-reward are set to $w^G = 0.5$ and $w^S = 0.5$ for all tasks. The character can be trained to perform tasks in a variety of distinct styles by providing the motion prior with different datasets. Figure 3 illustrates behaviors learned by the Humanoid on various tasks. Table 1 records the performance of the policies with respect to the normalized task return, and summary statistics of the different datasets used to train the motion priors are available in Table 2. AMP can accommodate large unstructured datasets, with the largest dataset containing 56 clips from 8 different human actors, for a total of 434s of motion data. In the case of the Target Heading task, a motion prior trained using a locomotion dataset, containing walking, running, and jogging motions, leads to a policy that executes different locomotion gaits depending on the target speed. Transitions between various gaits emerge automatically through the motion prior, with the character adopting walking gaits at slow speeds ($\sim 1$ m/s), switching to jogging gaits at faster speeds ($\sim 2.5$ m/s), and breaking into a fast run as the target speed approaches ($\sim 4.5$ m/s). The motion prior also leads to other human-like strategies, such as banking into turns, and slowing down before large changes in direction. The policies develop similar behaviors for the Target Location task. When the target is near the character, the policy adopts slower walking gaits. But when the target is further away, the character automatically transitions into a run. These intricate behaviors arise naturally from the motion prior, without requiring a motion planner to explicitly select which motion the character should execute in a given scenario, such as those used in prior systems [Bergamin et al. 2019; Luo et al. 2020;]
Table 1. Performance statistics of combining AMP with additional task objectives. Performance is recorded as the average normalized task return, with 0 being the minimum possible return per episode and 1 being the maximum possible return. The return is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. The motion prior can be trained with different datasets to produce policies that adopt distinct stylistic behaviors when performing a particular task.

<table>
<thead>
<tr>
<th>Character</th>
<th>Task</th>
<th>Dataset</th>
<th>Task Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humanoid</td>
<td>Target</td>
<td>Locomotion</td>
<td>0.90 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>Heading</td>
<td>Walk</td>
<td>0.46 ± 0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run</td>
<td>0.63 ± 0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stealthy</td>
<td>0.89 ± 0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zombie</td>
<td>0.94 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>Locomotion</td>
<td>0.63 ± 0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zombie</td>
<td>0.50 ± 0.00</td>
</tr>
<tr>
<td>Obstacles</td>
<td>Run + Leap + Roll</td>
<td>0.27 ± 0.10</td>
<td></td>
</tr>
<tr>
<td>Stepping</td>
<td>Cartwheel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stones</td>
<td>Jump</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dribble</td>
<td>Locomotion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strike</td>
<td>Walk + Punch</td>
<td>0.73 ± 0.02</td>
<td></td>
</tr>
<tr>
<td>T-Rex</td>
<td>Target</td>
<td>Locomotion</td>
<td>0.36 ± 0.03</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peng et al. 2017]. In addition to standard locomotion gaits, the motion prior can also be trained for more stylistic behaviors, such as walking like a shambling zombie or walking in a stealthy manner. Our framework enables the character to acquire these distinct styles by simply providing the motion prior with different unstructured motion datasets.

To determine whether the transitions between distinct gaits are a product of the motion prior or a result of the task objective, we train policies to perform the Target Heading task using limited datasets containing only walking or running data. Figure 4 compares the performance of policies trained with these different datasets. Policies trained with only walking motions learn to perform only walking gaits, and do not show any transitions to faster running gaits even at faster target speeds. As a result, these policies are not able to achieve the faster target speeds. Similarly, policies trained with only running motions are not able to match slower target speeds. Training the motion prior with a diverse dataset results in more flexible and optimal policies that are able to achieve a wider range of target speeds. This indicates that the diversity of behaviors exhibited by our policies can in large part be attributed to the motion prior, and is not solely a result of the task objective.

To further illustrate AMP’s ability to compose disparate skills, we introduce additional reference motions into the dataset for getting up from the ground in various configurations. These additional motion clips then enable our character to recover from a fall and continue to perform a given task (Figure 3(c)). The policy also discovers novel recovery behaviors that are not present in the dataset. When the character falls forward, it tucks its body into a roll during the fall in order to more quickly transition into a getup behavior. While this particular behavior is not present in the motion clips, the policy is able to generalize behaviors observed in the dataset to produce novel and naturalistic strategies for new scenarios.

For the Strike task (Figure 1), the motion prior is trained using a collection of walking motion clips and punching motion clips. The resulting policy learns to walk to the target when it is far away, and then transition to a punching motion once it is within range to hit the target. Note that the motion clips in the dataset contain strictly walking-only motions or punching-only motion, and none of the clips show an actor walking to and punching a target. Instead, the policy learns to temporally sequence these different behaviors in order to fulfill the high-level task objectives. Again, this composition of different skills emerges automatically from the motion prior, without requiring a motion planner or other mechanisms for motion selection.

Finally, our system can also train visuomotor policies for traversing obstacle-filled environments. By providing the motion prior with a collection of locomotion clips and rolling clips, the character learns to utilize these diverse behaviors to traverse the different obstacles. The character learns to leap over obstacles such as gaps. But as it approaches the overhead obstructions, the character transitions into a rolling behavior in order to pass underneath the obstacles. Previous systems that have demonstrated similar composition of diverse maneuvers for clearing obstacles have typically required a separate
motion planner or manual annotations [Liu et al. 2012; Park et al. 2019]. Our approach provides a unified framework where the same underlying algorithm is able to learn how to perform the various skills and which skill to execute in a given scenario. Furthermore, the character can also be trained to traverse obstacles in distinct styles by providing the motion prior with different motion clips, such as jumping or cartwheeling across stepping stones (Figure 3).

8.3 Comparisons

An alternative approach for learning a motion prior from unstructured motion data is to build a latent space model [Heess et al. 2016; Lynch et al. 2020; Merel et al. 2020; Peng et al. 2019a]. Unlike AMP, which encourages a character to adopt a desired motion style directly through an optimization objective, a latent space model enforces a particular motion style indirectly, by using a latent representation to constrain the policy’s actions to those that produce motions of the desired style. To compare AMP to these latent space models, we first pre-train a low-level controller using a motion tracking objective to imitate the same set of reference motions that are used to train the motion prior. The learned low-level controller is then used to train separate high-level controllers for each downstream task. Note that reference motions are used only during pre-training, and the high-level controllers are trained to optimize only the task objectives. A more in-depth description of the experimental setup for the latent space model is available in Appendix C.

A qualitative comparison of the behaviors learned using AMP and the latent space model is available in the supplementary video. Figure 5 compares the task performance of the different models, along with a baseline model trained from scratch for each task without leveraging any motion data. Both AMP and the latent space models are able to produce substantially more life-like behaviors than the baseline models. For the latent space models, since the low-level and high-level controllers are trained separately, it is possible for the distribution of encodings specified by the high-level controller to be different than the distribution of encodings observed by the low-level controller during pre-training [Luo et al. 2020]. This in turn can result in unnatural motions that deviate from the behaviors depicted in the original dataset. AMP enforces a motion style directly through the reward function, and is therefore able to better mitigate some of these artifacts. The more structured exploration behaviors from the latent space model enable the policies to solve downstream tasks more quickly. However, the pre-training stage used to construct the low-level controller can itself be sample intensive. In our experiments, the low-level controllers are trained using 300 million samples before being transferred to downstream tasks. With AMP, no such pre-training is necessary, and the motion prior can be trained jointly with the policy.

8.4 Single-Clip Imitation

Although our goal is to train characters with large motion datasets, to evaluate the effectiveness of our framework for imitating behaviors from motion clips, we consider a single-clip imitation task. In this setting, the character’s objective is to imitate a single motion clip at a time, without additional task objectives. Therefore,
policy is trained solely to maximize the style-reward $r_s^S$ from the motion prior. Unlike previous motion tracking methods, our approach does not require a manually designed tracking objective or a phase-based synchronization of the reference motion and the policy Peng et al. [2018a]. Table 3 summarizes the performance of policies trained using AMP to imitate a diverse corpus of motions. Figure 6 and 7 illustrate examples of motions learned by the characters. Performance is evaluated using the average pose error, where the pose error $e_t^{\text{pose}}$ at each time step $t$ is computed between the pose of the simulated character and the reference motion using the relative positions of each joint with respect to the root (in units of meters),

$$e_t^{\text{pose}} = \frac{1}{N_{\text{Joint}}} \sum_{j \in \text{joints}} \left\| (x_t^j - x_t^{\text{root}}) - (\dot{x}_t^j - \dot{x}_t^{\text{root}}) \right\|_2. \quad (10)$$

$x_t^j$ and $\dot{x}_t^j$ denote the 3D Cartesian position of joint $j$ from the simulated character and the reference motion, and $N_{\text{Joint}}$ is the total number of joints in the character’s body. This method of evaluating motion similarity has previously been reported to better conform to human perception of motion similarity [Harada et al. 2004; Tang et al. 2008]. Since AMP does not use a phase variable to synchronize the policy with the reference motion, the motions may progress at different rates, resulting in de-synchronization that can lead to large pose errors even when the overall motions are similar. To better evaluate the similarity of the motions, we first apply dynamic time warping (DTW) to align the reference motion with the motion of the simulated character [Sakoe and Chiba 1978], before computing the pose error between the two aligned motions. DTW is applied using Equation 10 as the cost function.

AMP is able to closely imitate a large variety of highly dynamic skills, while also avoiding many of the visual artifacts exhibited by prior adversarial motion imitation systems [Merel et al. 2017; Wang et al. 2017]. We compare the performance of our system to results produced by the motion tracking approach from Peng et al. [2018a], which uses a manually designed reward function and requires synchronization of the policy with a reference motion via a phase variable. Figure 8 compares the learning curves of the different methods. Since the tracking-based policies are synchronized with their respective reference motions, they are generally able to learn faster and achieve lower errors than policies trained with AMP. Nonetheless, our method is able to produce results of comparable quality without the need to manually design or tune reward functions for different motions. However, for some motions, such as the Front-Flip, AMP is prone to converging to locally optimal behaviors, where instead of performing a flip, the character learns to simply shuffle forwards in order to avoid falling. Tracking-based methods can mitigate these local optima by terminating an episode early if the character’s pose deviates too far from the reference motion [Peng et al. 2018a; Won et al. 2020]. However, this strategy is not directly applicable to AMP, since the policy is not synchronized with the reference motion. But as shown in the previous sections, this lack of synchronization is precisely what allows AMP to easily leverage large datasets of diverse motion clips to solve more complex tasks.

8.5 Ablations

Our system is able to produce substantially higher fidelity motions than prior adversarial learning frameworks for physics-based character control [Merel et al. 2017; Wang et al. 2017]. In this section, we identify critical design decisions that lead to more stable training and higher quality results. Figure 8 compares learning curves of policies trained on the single-clip imitation tasks with different components of the system disabled. Gradient penalty proves to be the most vital component. Models trained without this regularization tend to exhibit large performance fluctuations over the course of the training, and lead to noticeable visual artifacts in the final policies, as shown in the supplementary video. The addition of the gradient penalty not only improves stability during training, but also leads to substantially faster learning across a large set of skills. The inclusion of velocity features in the discriminator’s observations is also an important component for imitating some motions. In principle,
including consecutive poses as input to the discriminator should provide some information that can be used to infer the joint velocities. But we found that this was insufficient for some motions, such as rolling. As shown in the supplementary video, in the absence of velocity features, the character is prone to converging to a strategy of holding a fixed pose on the ground, instead of performing a roll. The additional velocity features are able to mitigate these dependencies. But we found that this was insufficient for some motions, such as rolling. As shown in the supplementary video, in the absence of velocity features, the character is prone to converging to a strategy of holding a fixed pose on the ground, instead of performing a roll. The additional velocity features are able to mitigate these dependencies.

### Table 3. Performance statistics of imitating individual motion clips without task objectives. 'Dataset Size' records the total length of motion data used for each skill. Performance is recorded as the average pose error (in units of meters) between the time-warped trajectories from the reference motion and simulated character. The pose error is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. Each episode has a maximum length of 20s. We compare our method (AMP) with the motion tracking approach proposed by Peng et al. [2018a]. AMP is able to closely imitate a diverse repertoire of complex motions, without manual reward engineering.

<table>
<thead>
<tr>
<th>Character</th>
<th>Motion</th>
<th>Dataset Size</th>
<th>Motion Tracking</th>
<th>AMP (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humanoid</td>
<td>Back-Flip</td>
<td>1.14s</td>
<td>0.070 ± 0.021</td>
<td>0.150 ± 0.028</td>
</tr>
<tr>
<td></td>
<td>Cartwheel</td>
<td>1.42s</td>
<td>0.039 ± 0.011</td>
<td>0.067 ± 0.014</td>
</tr>
<tr>
<td></td>
<td>Dance</td>
<td>1.80s</td>
<td>0.043 ± 0.001</td>
<td>0.099 ± 0.007</td>
</tr>
<tr>
<td></td>
<td>Front-Flip</td>
<td>1.60s</td>
<td>0.278 ± 0.040</td>
<td>0.425 ± 0.010</td>
</tr>
<tr>
<td></td>
<td>Jog</td>
<td>0.82s</td>
<td>0.029 ± 0.001</td>
<td>0.156 ± 0.001</td>
</tr>
<tr>
<td></td>
<td>Jump</td>
<td>1.77s</td>
<td>0.033 ± 0.001</td>
<td>0.083 ± 0.022</td>
</tr>
<tr>
<td></td>
<td>Roll</td>
<td>2.02s</td>
<td>0.072 ± 0.018</td>
<td>0.088 ± 0.008</td>
</tr>
<tr>
<td></td>
<td>Run</td>
<td>0.80s</td>
<td>0.028 ± 0.002</td>
<td>0.075 ± 0.015</td>
</tr>
<tr>
<td></td>
<td>Spin</td>
<td>1.00s</td>
<td>0.063 ± 0.022</td>
<td>0.094 ± 0.002</td>
</tr>
<tr>
<td></td>
<td>Side-Flip</td>
<td>2.41s</td>
<td>0.191 ± 0.043</td>
<td>0.124 ± 0.012</td>
</tr>
<tr>
<td></td>
<td>Spin-Kick</td>
<td>1.28s</td>
<td>0.042 ± 0.001</td>
<td>0.058 ± 0.012</td>
</tr>
<tr>
<td></td>
<td>Walk</td>
<td>1.39s</td>
<td>0.018 ± 0.005</td>
<td>0.030 ± 0.001</td>
</tr>
<tr>
<td></td>
<td>Zombie</td>
<td>1.68s</td>
<td>0.049 ± 0.013</td>
<td>0.058 ± 0.014</td>
</tr>
<tr>
<td>T-Rex</td>
<td>Turn</td>
<td>2.15s</td>
<td>0.098 ± 0.011</td>
<td>0.284 ± 0.023</td>
</tr>
<tr>
<td></td>
<td>Walk</td>
<td>2.00s</td>
<td>0.009 ± 0.005</td>
<td>0.096 ± 0.027</td>
</tr>
<tr>
<td>Dog</td>
<td>Canter</td>
<td>0.45s</td>
<td>0.026 ± 0.002</td>
<td>0.034 ± 0.002</td>
</tr>
<tr>
<td></td>
<td>Pace</td>
<td>0.63s</td>
<td>0.020 ± 0.001</td>
<td>0.024 ± 0.003</td>
</tr>
<tr>
<td></td>
<td>Spin</td>
<td>0.73s</td>
<td>0.026 ± 0.002</td>
<td>0.086 ± 0.008</td>
</tr>
<tr>
<td></td>
<td>Trot</td>
<td>0.52s</td>
<td>0.019 ± 0.001</td>
<td>0.026 ± 0.001</td>
</tr>
</tbody>
</table>

Fig. 8. Learning curves of various methods on the single-clip imitation tasks. We compare AMP to the motion tracking approach proposed by Peng et al. [2018a] (Motion Tracking), as well a version of AMP without velocity features for the discriminator (AMP - No Vel), and AMP without the gradient penalty regularizer (AMP - No GP). A comprehensive collection of learning curves for all skills are available in the Appendix. AMP produces results of comparable quality when compared to prior tracking-based methods, without requiring a manually designed reward function or synchronization between the policy and reference motion. Velocity features and gradient penalty are vital for effective and consistent results on challenging skills. However, like many other GAN-based techniques, AMP is susceptible to mode collapse. When provided with a large dataset of diverse motion clips, the policy is prone to imitating only a small subset of the example behaviors, ignoring other behaviors that may ultimately be more optimal for a given task. The motion priors in our experiments are also trained from scratch for each policy. But since the motion prior is largely task-agnostic, it should in principle be possible to transfer and reuse motion priors for different policies and tasks. Exploring techniques for developing general and transferable motion priors may lead to modular objective functions that can be conveniently incorporated into downstream tasks, without requiring retraining for each new task. While the motion prior does not require direct access to task-specific information, the data used to train the motion prior is generated by policies trained to perform a particular task. This may introduce some task dependencies into the motion prior, which can hinder its ability to be transferred to other tasks. Training motion priors using data generated from larger and more diverse repertoires of tasks may help to facilitate transferring the learned motion priors to new tasks. Our experiments also focus primarily on tasks that involve temporal composition of different skills, which require the character to perform different behaviors at different points in time. However, spatial composition might also be vital for some tasks that require a character to perform multiple skills simultaneously. Developing motion priors that are more amenable to spatial composition of disparate skills may lead to more flexible and sophisticated behaviors. Despite these limitations,
we hope this work provides a useful tool that enables physically simulated characters to take advantage of the large motion datasets that have been so effective for kinematic animation techniques, and open exciting directions for future exploration in data-driven physics-based character animation.

ACKNOWLEDGEMENTS
We thank Sony Interactive Entertainment for providing reference motion data for this project, Bonny Ho for narrating the video, the anonymous reviewers for their helpful feedback, and AWS for providing computational resources. This research was funded by an NSERC Postgraduate Scholarship, and a Berkeley Fellowship for Graduate Study.

REFERENCES


M. Al Borno, M. de Lasa, and A. Hertzmann. 2013. Trajectory Optimization for Full-Space Motion Data for this Project, Bonny Ho for Narrating the Video, the Anonymous Reviewers for Their Helpful Feedback, and AWS for Providing Computational Resources. This Research Was Funded by an NSERC Postgraduate Scholarship, and a Berkeley Fellowship for Graduate Study.

ACM
Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa


where \( g_t = (d_t^* \cdot \triangledown \phi) \), with \( d_t^* \) being the target direction in the character’s local coordinate frame. The task-reward is calculated according to:

\[
r_t^G = \exp \left(-0.25 (\phi^* - d_t^* \cdot \chi_{t}^{\text{com}})^2 \right) ,
\]

where \( \chi_t^{\text{com}} \) is the center-of-mass velocity of the character at time step \( t \), and the target speed is selected randomly between \( \phi^* \in [1, 5] \text{ m/s} \). For slower moving styles, such as Zombie and Stealthy, the target speed is fixed at \( 1 \text{ m/s} \).

**Target Location:** In this task, the character’s objective is to move to a target location \( x^* \). The goal \( g_t = \tilde{x}_t \) records the location in the character’s local coordinate frame. The task-reward is given by:

\[
r_t^G = 0.7 \exp \left(-0.5 ||x^* - \tilde{x}_t^{\text{root}}||^2 \right) + 0.3 \exp \left(-(\max(0, \phi^* - d_t^* \cdot \chi_{t}^{\text{com}}))^2 \right) .
\]

Here, \( \phi^* = 1 \text{ m/s} \) specifies a minimum target speed at which the character should move towards the target, and the character will not be penalized for moving faster than this threshold. \( d_t^* \) is a unit vector on the horizontal plane that points from the character’s root to the target.

**Dribbling:** To evaluate our system on more complex object manipulation tasks, we train policies for a dribbling task, where the objective is for the character to dribble a soccer ball to a target location. The reward function is given by:

\[
r_t^{G} = 0.1 r_t^{CV} + 0.1 r_t^{CP} + 0.3 r_t^{bv} + 0.5 r_t^{bp} \quad (13)
\]

\[
r_t^{CV} = \exp \left(-1.5 \max(0, \phi^* - d_t^{ball} \cdot \chi_{t}^{\text{com}})^2 \right) \quad (14)
\]

\[
r_t^{CP} = \exp \left(-0.5 ||d_t^{ball} - \chi_{t}^{\text{com}}||^2 \right) \quad (15)
\]

\[
r_t^{bv} = \exp \left(-\max(0, \phi^* - d_t^{ball} \cdot \chi_{t}^{\text{ball}})^2 \right) \quad (16)
\]

\[
r_t^{bp} = \exp \left(-0.5 ||\dot{x}_t - \chi_{t}^{\text{com}}||^2 \right) \quad (17)
\]

\( r_t^{CV} \) and \( r_t^{CP} \) encourage the character to move towards and stay near the ball, where \( x_t^{ball} \) and \( \dot{x}_t^{ball} \) represent the position and velocity of the ball, \( d_t^{ball} \) is a unit vector pointing from the character to the ball, and \( \phi^* = 1 \text{ m/s} \) is the target velocity at which the character should move towards the ball. Similarly, \( r_t^{bv} \) and \( r_t^{bp} \) encourages the character to move the ball to the target location, with \( d_t^{ball} \) denoting a unit vector pointing from the ball to the target. The goal \( g_t = \tilde{x}_t \) records the relative position of the target location with respect to the character. The state \( s_t \) is augmented with additional features that describe the state of the ball, including the position \( x_t^{ball} \), orientation \( \tilde{x}_t^{ball} \), linear velocity \( \dot{x}_t^{ball} \), and angular velocity \( \ddot{x}_t^{ball} \) of the ball in the character’s local coordinate frame.

**Strike:** Finally, to further demonstrate our approach’s ability to compose diverse behaviors, we consider a task where the character’s objective is to strike a target using a designated end-effector (e.g. hands). The target may be located at various distances from the character. Therefore, the character must first move close to the target before striking it. These distinct phases of the task entail different optimal behaviors, and thus requires the policy to compose and transition between the appropriate skills. The goal \( g_t = (\tilde{x}_t, h_t) \) records the location of the target \( \tilde{x}_t \) in the character’s local coordinate frame, along with an indicator variable \( h_t \) that specifies if the target has already been hit. The task-reward is partitioned into three phases:

\[
r_t^G = \begin{cases} 
1, & \text{target has been hit} \\
0.3 r_t^{near} + 0.3, & ||x^* - \tilde{x}_t^{\text{root}}|| < 1.375 m \ , \\
0.3 r_t^{far}, & \text{otherwise}. 
\end{cases}
\]

If the character is far from the target \( x^* \), \( r_t^{far} \) encourages the character to move to the target using a similar reward function as the Target Location task (Equation 12). Once the character is within a given distance of the target, \( r_t^{near} \) encourages the character to strike the target with a particular end-effector,

\[
r_t^{near} = 0.2 \exp \left(-2||x^* - \tilde{x}_t^{\text{eff}}||^2 \right) + 0.8 \exp \left(\frac{2}{3} d_t^{eff} \cdot \dot{x}_t^{eff}, 0, 1 \right) ,
\]

where \( \dot{x}_t^{eff} \) and \( \ddot{x}_t^{eff} \) denote the position and velocity of the end-effector, and \( d_t^{eff} \) is a unit vector pointing from the character’s root to the target. After striking the target, the character receives a constant reward of 1 for the remaining time steps.

**Obstacles:** Finally, we consider tasks that involve visual perception and interaction with more complex environments, where the character’s goal is to traverse an obstacle filled environment, while maintaining a target speed. Policies are trained for two types of environments. 1) An environment containing a combination of obstacles including gaps, steps, and overhead obstacles that the character must duck under. 2) An environment containing stepping stones that requires more precise contact planning. Examples of the environment are available in Figure 1 and 3. The task-reward is the same as the one used for the Target Heading task (Equation 11), except the target heading is fixed along the direction of forward progress. In order for the policy to perceive the upcoming obstacles, the state is augmented with a 1D height-field of the upcoming terrain. The height-field records the height of the terrain at 100 sample locations, uniformly spanning 10m ahead of the character.

**B AMP HYPERPARAMETERS**

Hyperparameter settings used in the AMP experiments are available in Table 4. For single-clip imitation tasks, we found that a smaller discount factor \( \gamma = 0.95 \) allows the character to more closely imitate a given reference motion. A larger discount factor \( \gamma = 0.99 \) is used for experiments that include additional task objective, since these tasks may require longer horizon planning, such as Dribble and Strike.
### C LATENT SPACE MODEL

The latent space model follows a similar architecture as Peng et al. [2019a] and Merel et al. [2019]. During pretraining, an encoder \( q(z_t|g_t) \) maps a goal \( g_t \) to a distribution over latent variables \( z_t \). A latent encoding \( z_t \sim q(z_t|g_t) \) is then sampled from the encoder distribution and passed to the policy as an input \( \pi(a_t|s_t, z_t) \). The latent distribution is modeled as a Gaussian distribution \( q(z_t|g_t) = \mathcal{N}(\mu_t(g_t), \Sigma_t(g_t)) \), with mean \( \mu_t(g_t) \) and diagonal covariance matrix \( \Sigma_t(g_t) \). The encoder is trained jointly with the policy using the following objective:

\[
\arg\max_{\pi, q} \mathbb{E}_{p(\tau|\pi,q)} \left[ \sum_{t=1}^{T-1} \gamma^t r_t \right] + \lambda \mathbb{E}_{p(g_t)} \left[ \text{D}_{\text{KL}} \left[q(\cdot|g_t)||p_0 \right] \right]. 
\]

\[\tau = (s_t, a_t, g_t, r_t)_{t=0}^{T-1}, s_T, g_T\] represents the goal-augmented trajectory, where the goal \( g_t \) may vary at each time step, and

\[
p(\tau|\pi,q) = p(g_0)p(s_0) \prod_{t=0}^{T-1} \left( g_{t+1}|g_t \right)p(s_{t+1}|s_t, a_t) \]

\[\int_{z_T} \pi(a_T|s_T, z_T)q(z_T|g_T)dz_T \]

is the likelihood of a trajectory under a given policy \( \pi \) and encoder \( q \). Similar to a VAE, we include a KL-regularizer with respect to a variational prior \( p_0(z_t) = \mathcal{N}(0, I) \) and coefficient \( \lambda \). The policy and encoder are trained end-to-end with PPO using the reparameterization trick [Kingma and Welling 2014]. Once trained, the latent space model can be transferred to downstream tasks by using \( \pi(a_t|s_t, z_t) \) as a low-level controller, and then training a separate high-level controller \( u(x_t|s_t, g_t) \) that specifies latent encodings \( z_t \) for the low-level controller. The parameters of \( \pi \) are fixed, and a new high-level controller \( u \) is trained for each downstream task.

During pretraining, the latent space model is trained using a motion imitation, where the objective is for the character to imitate a corpus of motion clips. A reference motion is selected randomly at the start of each episode, and a new reference motion is selected every 5-10s. The goal \( g_t = (q_{t+1}, q_{t+2}) \) specifies target poses from the reference motion at the next two time steps.

The networks used for \( \pi \) and \( u \) follow a similar architecture as the networks used for the policies trained with AMP. The encoder \( q \) is modeled by a network consisting of two hidden layers, with 512 and 256 hidden units, followed by a linear output layer for \( \mu_t(g_t) \) and \( \Sigma_t(g_t) \). The size of the latent encoding is set to 16D. Hyperparameter settings are available in Table 5.

### D SPATIAL COMPOSITION

Our experiments have so far focused primarily on temporal compositions of skills, where a character performs different skills at different points in time in order to fulfill particular task objectives, such as walking to a target and then punching it. In this section, we explore settings that require spatial composition of multiple skills, where the task requires a character to perform different skills simultaneously. To evaluate AMP in this setting, we consider a compositional task where a character needs to walk along a target heading direction while also waving its hand at a target height. The motion prior is trained using a dataset consisting of both walking motions and waving motions, but none of the motion clips show examples of walking and waving at the same time. Therefore, the onus is on the policy to spatially compose these different classes of skills in order to fulfill the two disparate objectives simultaneously.

In this task, the character has two objectives: 1) a target heading objective for moving along a target direction \( d^* \) at a target speed \( v^* \), and 2) a waving objective for raising its right hand to a target height \( y^* \). The goal input for the policy is given by \( g_t = (d^*_t, v^*, y^*) \), with \( d^*_t \) being the target direction in the character’s local coordinate frame. The composite reward is calculated according to:

\[
r_t^G = 0.5 r_t^{\text{heading}} + 0.5 r_t^{\text{waving}},
\]

Table 4. AMP hyperparameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w^a ) Task-Reward Weight</td>
<td>0.5</td>
</tr>
<tr>
<td>( w^s ) Style-Reward Weight</td>
<td>0.5</td>
</tr>
<tr>
<td>( w^K ) Gradient Penalty</td>
<td>10</td>
</tr>
<tr>
<td>Samples Per Update Iteration</td>
<td>4096</td>
</tr>
<tr>
<td>Batch Size</td>
<td>256</td>
</tr>
<tr>
<td>( \pi ) Policy Stepsize (Single-Clip Imitation)</td>
<td>( 2 \times 10^{-4} )</td>
</tr>
<tr>
<td>( \pi ) Policy Stepsize (Tasks)</td>
<td>( 4 \times 10^{-3} )</td>
</tr>
<tr>
<td>( V ) Value Stepsize (Single-Clip Imitation)</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>( V ) Value Stepsize (Tasks)</td>
<td>( 2 \times 10^{-5} )</td>
</tr>
<tr>
<td>( D ) Discriminator Stepsize</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>( B ) Discriminator Replay Buffer Size</td>
<td>( 10^9 )</td>
</tr>
<tr>
<td>( \gamma ) Discount (Single-Clip Imitation)</td>
<td>0.95</td>
</tr>
<tr>
<td>( \gamma ) Discount (Tasks)</td>
<td>0.99</td>
</tr>
<tr>
<td>SGD Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>GAE(\lambda)</td>
<td>0.95</td>
</tr>
<tr>
<td>TD(\lambda)</td>
<td>0.95</td>
</tr>
<tr>
<td>PPO Clip Threshold</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 5. Latent space model hyperparameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latent Encoding Dimension</td>
<td>16</td>
</tr>
<tr>
<td>( \lambda ) KL-Regularizer</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>Samples Per Update Iteration</td>
<td>4096</td>
</tr>
<tr>
<td>Batch Size</td>
<td>256</td>
</tr>
<tr>
<td>( \pi ) Policy Stepsize (Pre-Training)</td>
<td>( 2.5 \times 10^{-6} )</td>
</tr>
<tr>
<td>( u ) Policy Stepsize (Downstream Task)</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>( V ) Value Stepsize</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>( \gamma ) Discount (Pre-Training)</td>
<td>0.95</td>
</tr>
<tr>
<td>( \gamma ) Discount (Downstream Task)</td>
<td>0.99</td>
</tr>
<tr>
<td>SGD Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>GAE(\lambda)</td>
<td>0.95</td>
</tr>
<tr>
<td>TD(\lambda)</td>
<td>0.95</td>
</tr>
<tr>
<td>PPO Clip Threshold</td>
<td>0.02</td>
</tr>
</tbody>
</table>
where $r^{\text{heading}}_t$ the same as the reward used for the Target Heading task equation 11, and $r^{\text{wave}}_t$ is specified according to:

$$r^{\text{wave}}_t = \exp \left( -16 \left( y^{\text{hand}}_t - y^*_t \right)^2 \right),$$

(23)

where $y^{\text{hand}}_t$ is the height of character’s right hand.

To evaluate AMP’s ability to compose disparate skills spatially, we compare policies trained using both walking and waving motions, with policies trained with only walking motions or only waving motions. Table 6 compares the performance of the different policies with respect to the target heading and waving objectives. Although the motion prior was not trained with any reference motions that show both walking and waving at the same time, the policy was able to discover behaviors that combine these different skills, enabling the character to walk along different directions while also waving its hand at various heights. The policies trained with only walking motions tend to ignore the waving objective, and exhibit solely walking behaviors. Policies trained with only the waving motion are able to fulfill the waving objective, but learns a clumsy shuffling gait in order to follow the target heading direction. These results suggest that AMP does exhibit some capability for spatial composition different skills. However, the policies trained with both datasets can still exhibit some unnatural behaviors, particularly when the target height for the hand is high.

Table 6. Performance of policies trained using different datasets on a spatial compositional task that combines following a target heading and waving the character’s hand at a target height. The normalized task returns for each objective is averaged across 100 episodes for each model. The model trained with both walking and waving motions achieves relatively high rewards on both objectives, while the models trained with only one type of motions perform well only on one of the objectives.

<table>
<thead>
<tr>
<th>Dataset (Size)</th>
<th>Heading Return</th>
<th>Waving Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wave (51.7s)</td>
<td>0.683 ± 0.195</td>
<td>0.949 ± 0.144</td>
</tr>
<tr>
<td>Walk (229.7s)</td>
<td>0.945 ± 0.192</td>
<td>0.306 ± 0.378</td>
</tr>
<tr>
<td>Wave + Walk (281.4s)</td>
<td>0.885 ± 0.184</td>
<td>0.891 ± 0.202</td>
</tr>
</tbody>
</table>
Fig. 9. Learning curves comparing AMP to the motion tracking approach proposed by Peng et al. [2018a] (Motion Tracking) on the single-clip imitation tasks. Three policies initialized with different random seeds are trained for each method and motion. AMP produces results of comparable quality when compared to prior tracking-based methods, without requiring a manually designed reward function or synchronization between the policy and reference motion.
Fig. 10. Learning curves of applying AMP to various tasks and datasets.