PFPN: Continuous Control of Physically Simulated Characters using Particle Filtering Policy Network

Pei Xu
peix@clemson.edu
Clemson University
Charleston, South Carolina, USA

Ioannis Karamouzas
ioannis@clemson.edu
Clemson University
Charleston, South Carolina, USA

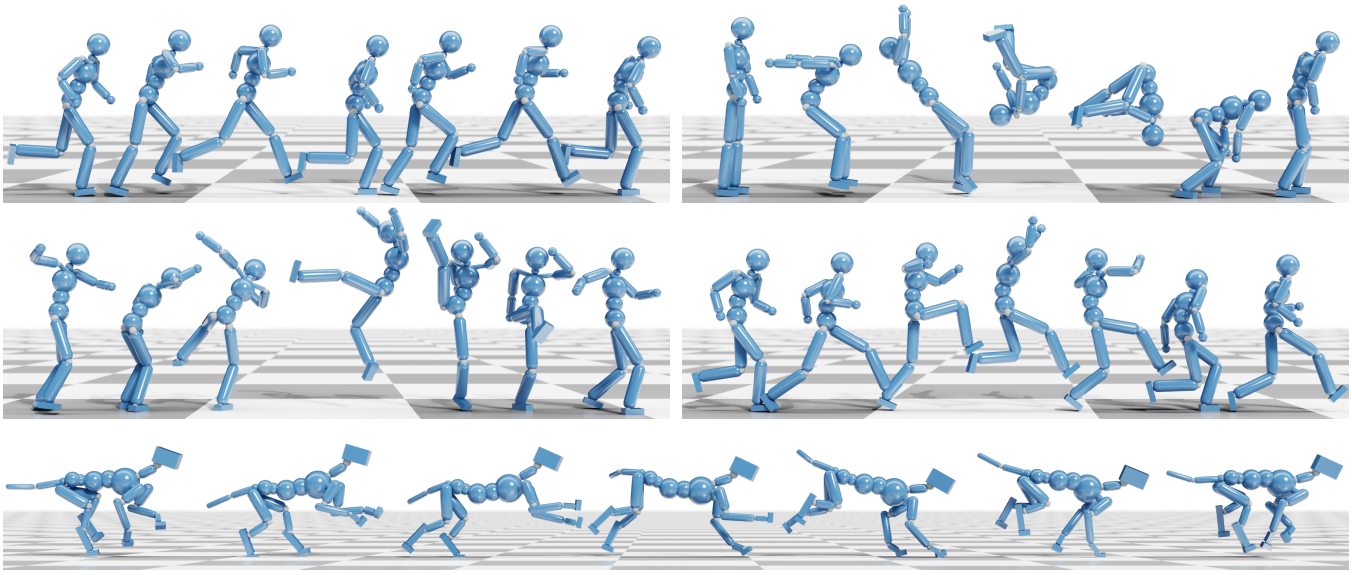


Figure 1: Motions generated through imitation learning using Particle Filtering Policy Network.

ABSTRACT

Data-driven methods for physics-based character control using reinforcement learning have been successfully applied to generate high-quality motions. However, existing approaches typically rely on Gaussian distributions to represent the action policy, which can prematurely commit to suboptimal actions when solving high-dimensional continuous control problems for highly-articulated characters. In this paper, to improve the learning performance of physics-based character controllers, we propose a framework that considers a particle-based action policy as a substitute for Gaussian policies. We exploit particle filtering to dynamically explore and discretize the action space, and track the posterior policy represented as a mixture distribution. The resulting policy can replace the unimodal Gaussian policy which has been the staple for character control problems, without changing the underlying model architecture of the reinforcement learning algorithm used to perform policy optimization. We demonstrate the applicability of our approach on various motion capture imitation tasks. Baselines using our particle-based policies achieve better imitation performance and speed of convergence as compared to corresponding implementations using Gaussians, and are more robust to external perturbations during character control. Related code is available at: `https://motion-lab.github.io/PFPN`.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MIG '21, November 10–12, 2021, Virtual Event, Switzerland
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9131-3/21/11…$15.00
https://doi.org/10.1145/3487983.3488301

CCS CONCEPTS

• **Computing methodologies** → **Animation**; *Physical simulation*; *Reinforcement learning*.

KEYWORDS

character animation, physics-based control, reinforcement learning

ACM Reference Format:
Pei Xu and Ioannis Karamouzas. 2021. PFPN: Continuous Control of Physically Simulated Characters using Particle Filtering Policy Network. In *Motion, Interaction and Games (MIG '21), November 10–12, 2021, Virtual Event, Switzerland*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3487983.3488301

1 INTRODUCTION

In the last few years, impressive results have been obtained for physics-based character control using data-driven methods under

the framework of deep reinforcement learning (DRL) [Bergamin et al. 2019; Chentanez et al. 2018; Ma et al. 2021; Merel et al. 2017; Peng et al. 2018; Won et al. 2020; Yin et al. 2021]. Works from [Jiang et al. 2019; Lee et al. 2019; Peng and van de Panne 2017; Reda et al. 2020] studied the learning and control performance with different action spaces using joint or muscle actuation formulations, and show the impact that action parameterization has on the trained policies. State-of-the-art approaches typically perform exploration in the action space of joint angles and control the character using PD servos to generate high-fidelity motions through imitation learning. Despite recent success, though, generating high-quality and robust animation for highly articulated characters is still a challenging task. Due to the infinite feasible action choices, controlling many degrees of is inherently ambiguous with respect to most behaviors, resulting in control problems that are under specified and highly dimensional.

State-of-the-art approaches for continuous character control define the action policy as a multivariate Gaussian distribution with independent components. Nevertheless, the unimodal form of a Gaussian distribution could prematurely commit to suboptimal actions when optimizing a reward function that consists of multiple competing terms and has a multimodal landscape [Hamalainen et al. 2020]. Consider, for example, an articulated character that needs to learn a mapping from states to actions based on all individual joints while tracking a reference motion. This is a very challenging task as for a given DOF and a continuous action space, we need to find a state-dependent mean and standard deviation. This process can be imagined as sliding the Gaussian to determine where to place it on an infinite line while also shrinking and swelling the distribution. And of course the problem becomes even more complicated as we need to do the same thing for all DOFs and determine how they work in synchrony for a given state.

To improve the policy expressivity beyond the unimodal Gaussian, recent works in character animation use primitive actions [Peng et al. 2019], coactivations [Ranganath et al. 2019], or a mixture of experts [Won et al. 2020] though the underlying distribution is still Gaussian. To address the unimodality issue of Gaussian policies, in the field of DRL, people have been exploring more expressive distributions than Gaussians, with a simple solution being to discretize the action space and use categorical distributions as multi-modal action policies [Andrychowicz et al. 2020; Jaśkowski et al. 2018; Tang and Agrawal 2019]. However, as we show, such a solution cannot scale well when training expert-guided control policies for physically simulated characters. The reason is that the performance of the action-space discretization depends a lot on the choice of discrete atomic actions, which are usually picked uniformly due to lack of prior knowledge. Therefore, a simple, fixed action discretization scheme typically can only provide suboptimal solutions that are unable to meet the fine control demands.

In this paper, we introduce an expressive, multimodal action policy for DRL-based learning of physics-based controllers for highly articulated characters. Following prior work, we employ a joint-actuation space and learn a mapping from states to joint target angles that are given as input to PD servos for computing the corresponding torque values. Instead of representing each action as a state-depending Gaussian, though, we exploit a particle approach to dynamically sample the action space during training and track the policy represented as a mixture of Gaussian distributions with state-independent components. We refer to the resulting policy network as Particle Filtering Policy Network (PFPN). We evaluate PFPN on benchmarks from the DeepMimic framework [Peng et al. 2018, 2020] involving motor control tasks for a humanoid and a dog character (see Figure 1 for some results). Our experiments show that baselines using PFPN exhibit better imitation performance and/or speed of convergence as compared to state-of-the-art Gaussian baselines, and lead to more robust character control under external perturbation. In addition, PFPN-trained controllers can generate motions with high visual quality for articulated characters performing motor tasks with the grace and naturalness of complex beings. Overall, PFPN offers a great alternative to Gaussian action policies leading to state-of-the-art controllers for physically simulated characters without introducing any notable computational overhead.

2 BACKGROUND

We consider a standard reinforcement learning setup where given a time horizon H and the trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, \cdots, \mathbf{s}_H, \mathbf{a}_H)$ obtained by a transition model $\mathcal{M}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and a parameterized action policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, with $\mathbf{s}_t$ and $\mathbf{a}_t$ denoting the state and action taken at time step t, respectively, the goal is to find the policy parameters θ that maximize the cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[r(\tau)\right] = \int p_\theta(\tau) r(\tau) d\tau. \tag{1}$$

Here, $p_\theta(\tau)$ denotes the state-action visitation distribution for the trajectory τ induced by the transient model $\mathcal{M}$ and the action policy π_θ, and $r(\tau) = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$ where $r(\mathbf{s}_t, \mathbf{a}_t)$ is the reward received at time step t. We can maximize $J(\theta)$ by adjusting the policy parameters θ through the gradient ascent method, where the gradient of the expected reward can be determined according to the policy gradient theorem [Sutton et al. 2000], i.e.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|\mathbf{s}_t)}\left[A_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)|\mathbf{s}_t\right], \tag{2}$$

where A_t denotes an estimate to the reward term $r_t(\tau)$. In DRL, the estimator of A_t often relies on a separate network (critic) that is updated in tandem with the policy network (actor). This gives rise to a family of policy gradient algorithms known as actor-critic.

Given a multi-dimensional continuous action space, the most common choice in current DRL baselines is to model the policy π_θ as a multivariate Gaussian distribution with independent components for each action dimension. For simplicity, let us consider a simple case with a single action dimension and define the action policy as $\pi_\theta(\cdot|\mathbf{s}_t) := \mathcal{N}(\mu_\theta(\mathbf{s}_t), \sigma^2_\theta(\mathbf{s}_t))$. Then, we can obtain $\log \pi_\theta(a_t|\mathbf{s}_t) \propto -(a_t - \mu_\theta(\mathbf{s}_t))^2$. Given a sampled action a_t and the estimate of cumulative rewards A_t, the optimization process based on the above expression can be imagined as that of shifting $\mu_\theta(\mathbf{s}_t)$ towards the direction of a_t if A_t is higher than the expectation, or to the opposite direction if A_t is smaller. Such an approach, though, can easily converge to a suboptimal solution, if, for example, the reward landscape has a basis between the current location of $\mu_\theta(\mathbf{s}_t)$ and the optimal solution, or hard to be optimized if the reward landscape is symmetric around $\mu_\theta(\mathbf{s}_t)$. These issues arise due to the fact that the Gaussian distribution is inherently unimodal, while the reward landscape could be multi-modal [Haarnoja et al. 2017].

We refer to Appendix A for further discussion about the limitations of unimodal Gaussian policies and the value of expressive multimodal policies. Indeed, Hamalainen et al. [2020] showed that the reward landscape for high-dimensional character control tasks typically has a complex shape and is often multimodal. This means that Gaussian policies may face difficulties during optimization for character control.

3 PARTICLE FILTERING POLICY NETWORK

In this section, we describe our Particle Filtering Policy Network (PFPN) that addresses the unimodality issues from which typical Gaussian-based policy networks suffer. Our approach represents the action policy as a mixture distribution obtained by adaptively discretizing the action space using state-independent particles. The policy network, instead of directly generating actions, it is tasked with choosing particles, while the final actions are obtained by sampling from the selected particles.

3.1 Particle-Based Action Policy

We define $\mathcal{P} := \{\langle \mu_{i,k}, w_{i,k}(\mathbf{s}_t|\theta)\rangle | i = 1,\dots,n; k = 1,\dots,m\}$ as a weighted set of particles for continuous control problems having an m-dimensional action space and n particles distributed on each action action space. Here, $\mu_{i,k}$ represents an atomic action location on the k-th dimension of the action space, and $w_{i,k}(\mathbf{s}_t|\theta)$ denotes the associated weight generated by the policy network with parameters θ given the input state $\mathbf{s}_t$. Let $p_{i,k}(a_{i,k}|\mu_{i,k}, \xi_{i,k})$ denote the probability density function of the distribution defined by the location $\mu_{i,k}$ and a stochastic process $\xi_{i,k}$ for sampling. Given $\mathcal{P}$, we define the action policy as factorized across the action dimensions:

$$\pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \prod_k \sum_i w_{i,k}(\mathbf{s}_t|\theta)\; p_{i,k}(a_{t,k}|\,\mu_{i,k},\,\xi_{i,k}), \tag{3}$$

where $\mathbf{a}_t = \{a_{t,1}, \cdots, a_{t,m}\}$, $a_{t,k}$ is the sampled action at the time step t for the action dimension k, and $w_{i,k}(\cdot|\theta)$ is obtained by applying a softmax operation to the output of the policy network for the k-th dimension and satisfies $\sum_i w_{i,k} = 1$. The state-independent parameter set, $\{\mu_{i,k}\}$, gives us an adaptive discretization scheme that can be optimized during training. The noise parameter, $\xi_{i,k}$, provides a way to generate stochastic actions during policy training. In our implementation, we choose Gaussian distributions with a standard deviation of $\xi_{i,k}$ for action sampling during training and then each particle can be regarded as a state-independent Gaussian of $\mathcal{N}(\mu_{i,k}, \xi_{i,k}^2)$. Without loss of generality, we define the parameters of a particle as $\phi_{i,k} = [\mu_{i,k}, \xi_{i,k}]$ for the following discussion.

While the softmax operation gives us a categorical distribution defined by $w_{\cdot,k}(\mathbf{s}_t|\theta)$, the nature of the policy for each dimension is a mixture distribution with state-independent components defined by $\phi_{i,k}$. The number of output neurons in PFPN increases linearly as the number of action dimensions increases, and thus makes it suitable for high-dimensional control problems. Drawing samples from the mixture distribution can be done in two steps. first, based on the weights $w_{\cdot,k}(\mathbf{s}_t|\theta)$, we perform sampling on the categorical distribution to choose a particle j_k for each dimension k, i.e.

$$j_k(\mathbf{s}_t) \sim P(\cdot|w_{\cdot,k}(\mathbf{s}_t)). \tag{4}$$

Then, we can draw actions from the components represented by the chosen particles with noise as

$$a_{t,k} \sim p_{j_k(\mathbf{s}_t)}(\cdot|\phi_{j_k(\mathbf{s}_t)}). \tag{5}$$

3.2 Training

The proposed particle-based policy distribution is general and can be applied directly to any algorithm using the policy gradient method with Equation 2. To initialize the training, due to lack of prior knowledge, the particles can be distributed uniformly along the action dimensions with a standard deviation covering the gap between two successive particles. With no loss of generality, let us consider below only one action dimension and drop the subscript k. Then, at every training step, each particle i will move along its action dimension and be updated by

$$\nabla J(\phi_i) = \mathbb{E}\left[\sum_t c_t w_i(s_t|\theta)\nabla_{\phi_i} p_i(a_t|\phi_i)|\mathbf{s}_t\right] \tag{6}$$

where $a_t \sim \pi_\theta^{\mathcal{P}}(\cdot|\mathbf{s}_t)$ is the action chosen during sampling, and $c_t = \dfrac{A_t}{\sum_j w_j(\mathbf{s}_t|\theta)p_j(a_t|\phi_j)}$ is a coefficient shared by all particles on the same action dimension. Our approach focuses only on the action policy representation in general policy gradient methods. The estimation of A_t can be chosen as required by the underlying policy gradient method, e.g. the generalized advantage estimator [Schulman et al. 2015] in PPO/DPPO. Similarly, for the update of the policy neural network, we have

$$\nabla J(\theta) = \mathbb{E}\left[\sum_t c_t p_i(a_t|\phi_i)\nabla_\theta w_i(\mathbf{s}_t|\theta)|\mathbf{s}_t\right]. \tag{7}$$

From the above equations, although sampling is performed on only one particle for each given dimension, all of that dimension's particles will be updated during each training iteration to move towards or away from the location of a_t according to A_t. The amount of the update, however, is regulated by the state-dependent weight $w_i(\mathbf{s}_t|\theta)$: particles that have small probabilities to be chosen for a given state $\mathbf{s}_t$ will be considered as uninteresting and be updated with a smaller step size or not be updated at all. On the other hand, the update of weights is limited by the distance between a particle and the sampled action: particles too far away from the sampled action would be considered as insignificant to merit any weight gain or loss. In summary, particles can converge to different optimal locations near them during training and be distributed multimodally according to the reward landscape defined by A_t, rather than collapsing to a unimodal, Gaussian-like distribution.

3.3 Resampling

Similar to traditional particle filtering approaches, our approach would encounter the problem of degeneracy [Kong et al. 1994]. During training, a particle placed near a location at which sampling gives a low A_t value would achieve a weight decrease. Once the associated weight reaches near zero, the particle will not be updated anymore (cf. Equation 6) and become 'dead'. We adapt the idea of resampling from the particle filtering literature [Doucet et al. 2001] to perform resampling for dead particles and reactivate them by duplicating alive target particles.

Algorithm 1: Policy Gradient Method using PFPN

Initialize the neural network parameter θ and learning rate α;
initialize particle parameters ϕ_i to uniformly distribute particles on each action dimension;
initialize the threshold ϵ to detect dead particles;
initialize the value of interval n to perform resampling.
while *training does not converge* **do**
  for *each environment step* **do**
    // Record the weight while sampling.
    $a_t \sim \pi_{\theta,\mathcal{P}}(\cdot|s_t)$; $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{w_i(s_t|\theta)\}$
  end
  for *each training step* **do**
    // Update parameters using SGD method.
    $\phi_i \leftarrow \phi_i + \alpha \nabla J(\phi_i)$ *// Equation 6*
    $\theta \leftarrow \theta + \alpha \nabla J(\theta)$ *// Equation 7*
  end
  for *every n environment steps* **do**
    // Detect dead particles and set up target ones.
    for *each particle i* **do**
      if $\max_{w_i \in \mathcal{W}_i} w_i < \epsilon$ **then**
        $\tau_i \sim P(\cdot|\mathbb{I}[w_k|w_k \in \mathcal{W}_k], k = 1, 2, \cdots)$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_i\}$; $\mathcal{D}_{\tau_i} \leftarrow \mathcal{D}_{\tau_i} \cup \{i\}$
      end
    end
    // Resampling.
    for *each target particle* $\tau \in \mathcal{T}$ **do**
      for *each dead particle* $i \in \mathcal{D}_\tau$ **do**
        // Duplicate particles.
        $\phi_i \leftarrow \phi_\tau$ with $\mu_i \leftarrow \mu_\tau + \varepsilon_i$
        // Duplicate parameters of the last layer in the policy network.
        $\omega_i \leftarrow \omega_\tau$; $b_i \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1)$
      end
      $b_\tau \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1)$; $\mathcal{D}_\tau \leftarrow \emptyset$
    end
    $\mathcal{T} \leftarrow \emptyset$; $\mathcal{W}_i \leftarrow \emptyset$
  end
end

A particle is considered dead if its maximum weight over all possible states is too small, i.e.

$$\max_{\mathbf{s}_t} w_i(\mathbf{s}_t|\theta) < \epsilon \tag{8}$$

where ϵ is a small positive threshold number. In practice, we cannot check $w_i(s_t|\theta)$ for all possible states, but can keep tracking it during sampling based on the observed states collected in the last batch of environment steps. During resampling, a target particle τ_i is drawn for each dead particle i independently. We consider two resampling strategies: (1) the unweighted resampling strategy that picks a τ_i randomly from all alive particles; and (2) the weighted resampling strategy that draws a target τ_i from the categorical distribution obtained from the average weight of each particle over the observed samples, i.e.

$$\tau_i \sim P\left(\cdot|\mathbb{E}_{\mathbf{s}_t}\left[w_k(\mathbf{s}_t|\theta)\right], k = 1, 2, \cdots\right). \tag{9}$$

Both of these two resampling strategies are stochastic. We, by default, use the weighted resampling, which considers the importance of the target particle candidates leading to a more stable learning performance, as we will show empirically in Section 4.5.

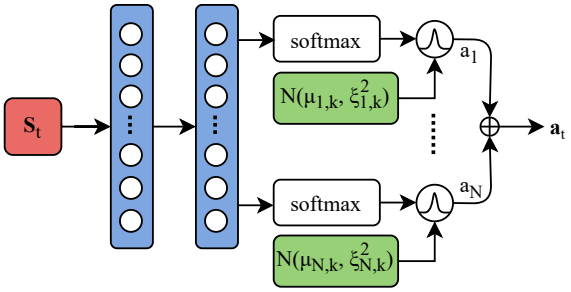


Figure 2: PFPN architecture with a N-dimension action space in our experiment. Each particle represents a state-independent Gaussian distribution of $\mathcal{N}(\mu_{i,k}, \xi^2_{i,k})$ where $k = 1, \cdots, 35$ for 35 particles on each action dimension and $i = 1, \cdots, N$. $\oplus$ denotes the concatenation operator.

Theorem. *Let $\mathcal{D}_\tau$ be a set of dead particles sharing the same target particle τ. Let also the logits for the weight of each particle k be generated by a fully-connected layer with parameters $\boldsymbol{\omega}_k$ for the weight and b_k for the bias. The policy $\pi^{\mathcal{P}}_\theta(a_t|\mathbf{s}_t)$ is guaranteed to remain unchanged after resampling via duplicating $\phi_i \leftarrow \phi_\tau, \forall i \in D_\tau$, if the weight and bias used to generate the unnormalized logits of the target particle are shared with those of the dead one as follows:*

$$\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau; \quad b_i, b_\tau \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1). \tag{10}$$

Proof. See Appendix D for the inference. □

The theorem guarantees the correctness of our resampling process as it keep the action policy $\pi^{\mathcal{P}}_\theta(a_t|\mathbf{s}_t)$ identical before and after resampling. If, however, two particles are exactly the same after resampling, they will always be updated together at the same pace during training and lose diversity. To address this issue, we add some regularization noise to the mean value when performing resampling, i.e. $\mu_i \leftarrow \mu_\tau + \varepsilon_i$, where ε_i is a small random number to prevent μ_i from being too close to its target μ_τ. We refer to Algorithm 1 for the outline of our proposed PFPN approach.

3.4 Action-Value Based Optimization

Algorithm 1 can be applied on general policy gradient algorithms, like PPO [Schulman et al. 2017] and A3C [Mnih et al. 2016]. However, we note that a number of algorithms, such as DDPG [Lillicrap et al. 2015], SAC [Haarnoja et al. 2018a,b] and their variants [Fujimoto et al. 2018; Haarnoja et al. 2017], perform optimization by maximizing a soft state-action value $Q(\mathbf{s}_t, \mathbf{a}_t)$. In this case, the action policy is required to be reparameterizable such that the sampled action $\mathbf{a}_t$ can be rewritten as a function differentiable to the policy network parameter θ, and the optimization can be done through the gradient $\nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t) \nabla_\theta \mathbf{a}_t$.

Our two-step sampling method for PFPN described in Section 3.1 is non-reparameterizable, because of the standard way of sampling from the categorical distribution through which Gaussians are mixed. To address this issue and enable the proposed action policy applicable in state-action value based off-policy algorithms, we consider the concrete distribution [Jang et al. 2016; Maddison

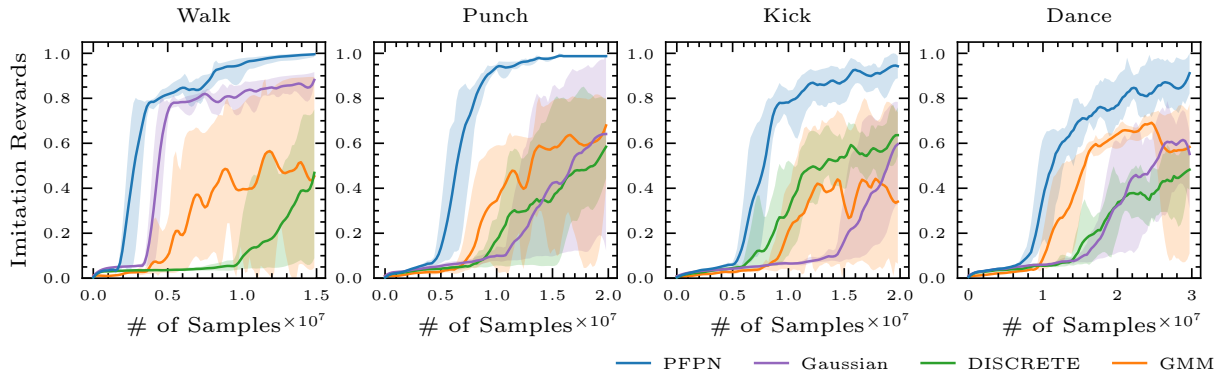


Figure 3: Learning curves of PFPN compared to other baselines using DPPO algorithm on humanoid character control tasks. Solid lines report the average and shaded regions are the minimum and maximum imitation rewards achieved with different random seeds during training.

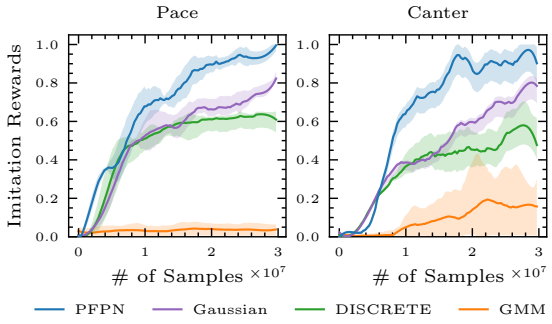


Figure 4: Learning performance of baselines using DPPO for dog character control.

et al. 2016] that generates a reparameterized continuous approximation to a categorical distribution. We refer to Appendix F for the reprarameterization trick and further details on the application of PFPN to off-policy algorithms.

4 EXPERIMENTS

The goal of our experiments is to evaluate whether PFPN can outperform the corresponding implementations with Gaussian policies in data-driven control tasks of physics-based character. We focus on the two aspects of comparisons: (1) the imitation ability of the trained policies, and (2) the policy robustness facing external force perturbation. We also perform sensitivity analysis on the learning performance of PFPN with different number of particles and resampling strategies.

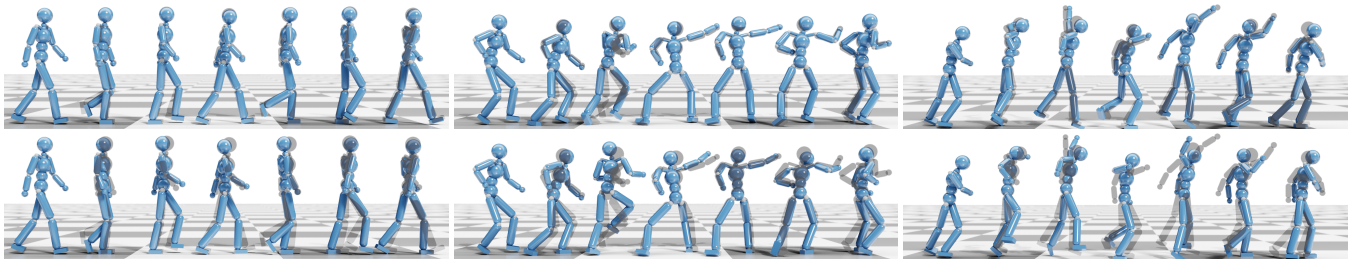
4.1 Setup

We run benchmarks based on the DeepMimic framework [Peng et al. 2018], which is the state-of-the-art DRL imitation learning framework for physics-based character control. The simulated character is controlled through stable proportional derivative controllers [Tan et al. 2011] running with the forward dynamic simulation at 600 Hz, while the policy network provides the control signal at 30 Hz. Following the settings of DeepMimic, we use the link position, orientation (in the unit of quaternion) and linear and angular velocity related to the root link position and heading direction, adding a phase variable to indicate the target pose implicitly at each time step and a variable indicating the root link height, as the observation space. We considered two articulated characters, a humanoid and a dog. The humanoid character has 8 spherical joints and 4 revolute joints, plus a root link and two end-effectors (hands) connected to the forearms with fixed joints, resulting in an observation space of $\mathbb{R}^{197}$ and action space of $\mathbb{R}^{36}$. The dog character has 18 spherical joints and 4 revolute joints, which lead an observation space of $\mathbb{R}^{301}$ and action space of $\mathbb{R}^{76}$. All simulations are run using PyBullet [Coumans and Bai 2021].

PFPN focuses only on the distribution for action policy and can be applied with any general policy-gradient DRL algorithm. In the following, we evaluate PFPN with policies trained using DPPO [Heess et al. 2017] with surrogate policy loss [Schulman et al. 2017] (We refer to Appendix C for results obtained with the A3C [Mnih et al. 2016] and IMPALA algorithms [Espeholt et al. 2018], and to Appendix B for all hyperparameters used). In our DPPO implementation, policy and value networks have a similar structure of two hidden fully-connected (FC) layers with neurons of 1024 and 512 respectively, as shown in Figure 2. The input state is normalized by moving average that is dynamically updated during training. By default, we place 35 particles on each action dimension and use the set of particles as a mixture of Gaussians with state-dependent weights but state-independent components.

4.2 Baseline Comparison

In Figure 3 and 4, we compare PFPN to baselines using Gaussian distribution with state-dependent mean and standard deviation values on the humanoid and dog character control tasks respectively. The imitation performance is measured in the term of normalized cumulative rewards of evaluation rollouts during training. We also compare PFPN to a fixed discretization scheme (DISCRETE) obtained by uniformly discretizing each action dimension into a fixed number of bins and sampling actions from a categorical distribution, and to policies using the distribution of a fully state-dependent



(a) Walk (b) Punch (c) Dance

Figure 5: Qualitative comparisons of the motions generated by PFPN (top) and Gaussian (bottom) baselines. Shadow characters indicate the reference motion.

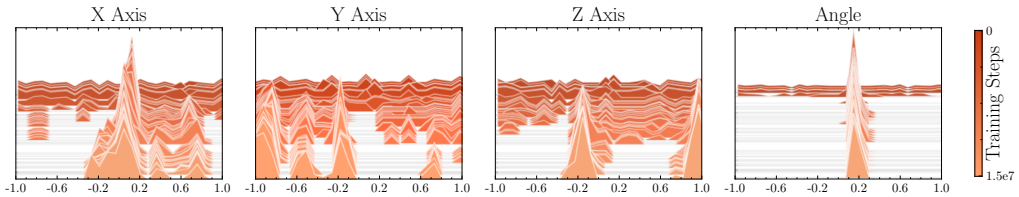


Figure 6: Evolution of how particles along the four action dimensions of the right hip joint are distributed during training of the Walk task. Each action dimension is normalized between -1 and 1. Particles are initially distributed uniformly along a dimension (dark colors) and their locations adaptively change as the policy network is trained (light colors). The training steps are measured by the number of samples exploited during training.

Gaussian Mixture Model (GMM). All baselines use the same network architecture with the same number of hidden neurons. PFPN, DISCRETE and GMM also have the same number of atomic actions (35 particles/bins) at each action dimension. We train five trials of each baseline with different random seeds that are the same across PFPN and the corresponding implementations of other methods. Evaluation was performed ten times every 1,000 training steps using deterministic actions.

As it can be seen from the figures, PFPN outperforms other baselines in all of the tested tasks. Our particle-based scheme achieves better final performance and exhibits faster convergence while being more stable across multiple trials. GMM uses Gaussian components with state-dependent mean and standard deviation values, and does not work better than Gaussian baselines. DISCRETE discretizes the action space using the atomic actions, which are the same with the initial distribution of the particles employed by PFPN. Though both DISCRETE and PFPN use the state-independent atomic action settings, PFPN optimizes the distribution of atomic actions represented by particles during training, and the resulting adaptive discretization scheme leads to higher asymptotic performance. Intuitively, by introducing more atomic actions, DISCRETE and PFPN could achieve higher control capacity. However, the more atomic actions employed the harder the optimization problem will be due to the increase of policy gradient variance [Tang and Agrawal 2019] (see Appendix E for theoretical analysis). In theory, GMM is a more general case of PFPN without resampling. However, we found that GMM does not usually work quite well and can perform even worse than Gaussians. This is consistent with the results reported by Tang and Agrawal [2018] for torque-based locomotion

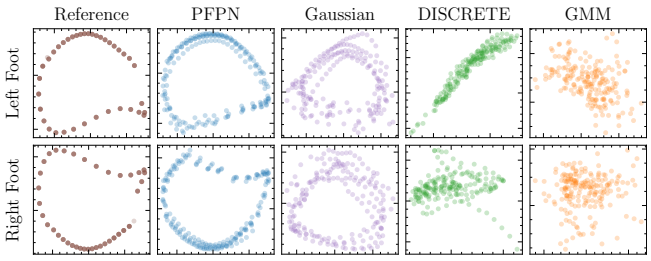


Figure 7: PCA embedding of the character's foot trajectories related to the base link position during ten gait cycles of walking. The reference one is one-gait walking motion obtained through motion capture.

control tasks. A possible reason is that GMM needs a much larger policy network. This may pose a challenge to the optimization. Another issue of GMMs observed during our experiments is that the Gaussian components are easy to collapse together and lose the advantage of multimodality.

To gain a better understanding of the advantages of using PFPN for learning motor tasks, Figure 6 shows how particles evolve during training for one of the humanoid's joints in the "Walk" task. We can see that each of the final active action spaces follows a multimodal distribution that covers only some small parts of the entire action space. PFPN optimizes the placement of atomic actions, providing an effective discretization scheme that reaches better performance compared to other baselines.

4.3 Motion Quality

In Figure 7, we compare the foot trajectories of each baseline during ten gait cycles in "Walk" tasks. From the figure, GMM generates jittery motions with unstable foot trajectories. DISCRETE can provide relatively stable trajectories, in which, however, the character moves always towards an inclined direction. Gaussian gives gaits with a cyclic pattern but not quite stable. PFPN generate stable gaits with a clear cyclic pattern closely following the reference motion. In Figure 5, we qualitatively compare the motions generated by PFPN and Gaussian baselines. As can be seen, the motion generated by PFPN follows the reference motion (shadow character) closely. Gaussian, though providing a human-like walking motion, exhibits visual artifacts in the other, more complex tasks, e.g. foot sliding and character drifting during dancing. Similar conclusions can be drawn for the dog character. When a fixed number of samples are exploited for training as reported in Figure 4, PFPN baselines can track the reference motion closely while Gaussian baselines perform worse, with the dog, for example, having evident jerky movements during pacing.

In addition, we note that characters trained with Gaussian-based policies often lack the grace seen in living beings as compared to PFPN-trained characters that exhibit behavior more close to the one seen in the reference data. For example, while the dog is able to achieve a relatively high reward according to DeepMimic's tracking reward function (Figure 4), its tail moves in an unnatural way during canter. Similarly, while this is more subtle, the Gaussian-based humanoid character stamps its feet on the ground while walking, resulting in a heavy-footed gait as compared to the graceful gait of the PFPN-based humanoid. We refer to the supplemental videos for more details on the visual quality of the trained controllers.

4.4 Robustness

We evaluate policy robustness through projectile testing and also external force perturbations. During projectile testing, the character is controlled by the "walk" policy, and a cube projectile with side length of 0.2m and initial velocity of 0.2m/s is cast towards the torso of the character. Figure 8 reports the number of frames that a policy can control the character before falling down on the ground while varying the mass of the cube. In Table 1, we also report the minimal force needed to push the character down in "walk" and "punch" tasks for the humanoid, and in "pace" and "canter" tasks for the dog character. All experiments were performed after training using deterministic actions. It is evident that the character controlled by PFPN are more robust to external disturbances and can sustain much higher forces than other baselines.

4.5 Ablation Study

Resampling Strategy. In Figure 9a, we compare PFPN with default, weighted resampling to PFPN with unweighted resampling (see Section 3.3), and to PFPN without any resampling. The weighted resampling strategy draws targets for dead particles according to the weights of the remaining, alive ones. The unweighted resampling strategy draws targets uniformly from alive particles. It can be seen that both weighted and unweighted resampling could help improve the training performance significantly. However, unweighted resampling could lead to high variance by introducing too much uncertainty, since it could reactivate dead particles and place them in suboptimal locations with higher probability, compared to the weighted resampling. After resampling, even though the particles would be optimized or resampled once more if they are placed in bad locations, this could make the training process converge slower, as shown in the figure.

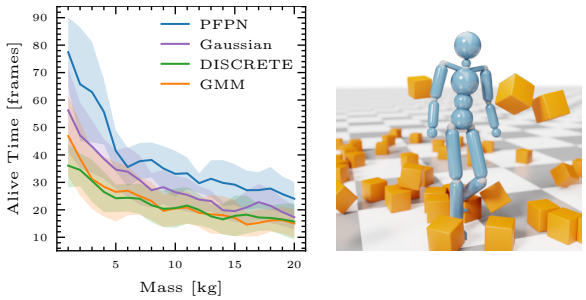
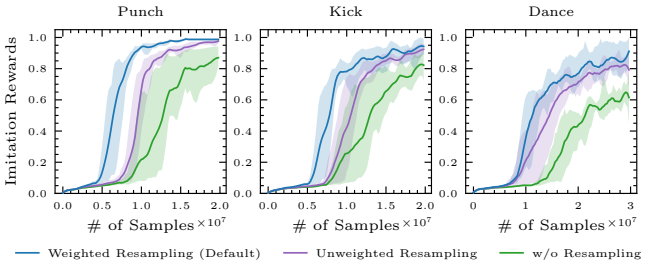


Figure 8: Policy robustness of "Walk" motion with cube projectiles of varying mass. Solid lines report the average performance over ten trials and shaded regions indicate the standard deviation. All policies are obtained using DPPO algorithms without projectile training. Right: a character under projectile testing.

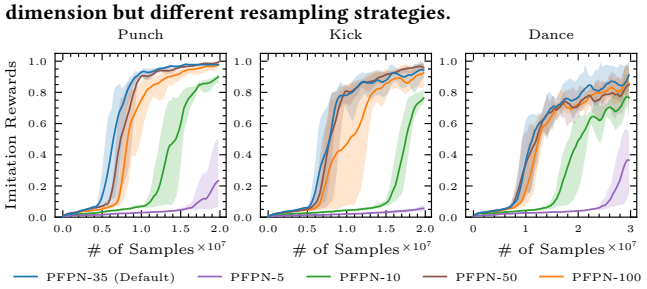
Table 1: Minimal forward and sideways push needed to make the character fall down. Push force is measured in Newtons (N) and applied on the chest of the character for 0.1s. The DISCRETE and GMM results are skipped for the dog character, as the respective trained controllers are unable to make the character walk even in the absence of external disturbances.

Task	Force Direction	PFPN	Gaussian	DISCRETE	GMM
Humanoid					
Walk	Forward	**588**	512	340	373
	Sideway	**602**	560	420	417
Punch	Forward	**1156**	720	480	721
	Sideway	**896**	748	576	732
Dog					
Pace	Forward	**735**	672	-	-
	Sideway	**412**	276	-	-
Canter	Forward	**418**	404	-	-
	Sideway	**344**	274	-	-

Number of Particles. Since the particle configuration in PFPN is state-independent, it needs a sufficient number of particles to meet the fine control demand. Intuitively, employing more particles will increase the resolution of the action space, and thus increase the control capacity and make fine control more possible. However, in Appendix E, we prove that due to the variance of policy gradient increasing as the number of particles increases, the more particles employed, the harder the optimization would be. Therefore, it may negatively influence the performance to employ too many particles. This conclusion is consistent with the results shown in Figure 9b.



(a) Learning performance of PFPN with 35 particles on each action dimension but different resampling strategies.



(b) Comparison of PFPN using 35 particles per action dimension (PFPN-35) to that using 5, 10, 50 and 100 particles.

Figure 9: Sensitivity of PFPN to different resampling strategies and the number of particles.

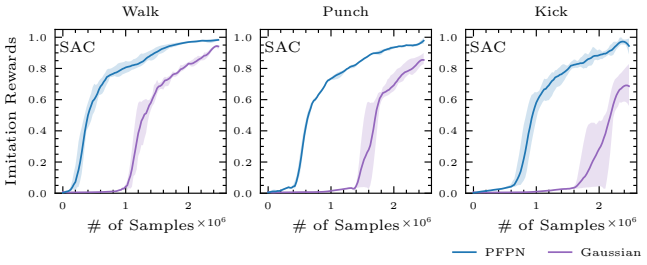


Figure 10: Learning performance of SAC with PFPN and Gaussian baselines.

As it can be seen, PFPN with 5 and 10 particles per action dimension performs badly; though using 50 or 100 particles per action dimension slows down the convergence speed a little bit, our approach is not sensitive in terms of the final learning performance when a relative large number of particles are employed.

4.6 PFPN Results with SAC

In this section, we highlight PFPN's performance in state-of-the-art off-policy DRL algorithm of SAC [Haarnoja et al. 2018b]. As shown in Figure 10, PFPN outperforms Gaussian baselines with faster convergence speed in all the tested tasks using SAC. While SAC has been successfully explored for continuous control problems in the machine learning community, DPPO is still the most commonly used DRL algorithm for training physics-based character controllers in the animation field. During experiments, we found that PFPN with SAC is more stable in terms of learning performance and more sampling efficient compared to DPPO implementations as reported in Figure 3. We refer to the supplementary video for comparisons between SAC and DPPO. Overall, PFPN with SAC needs only 2.5-million samples to train high-quality humanoid controllers in DeepMimic tasks as compared to DPPO that typically requires around 20 millions or even more samples to achieve the similar performance.

5 DISCUSSION AND FUTURE WORK

We present PFPN as a general framework for systematic exploration of high-dimensional action spaces during training of physics-based character controllers. Our approach uses a mixture of state-independent Gaussians represented by a set of weighted particles to track the action policy, as opposite to the multivariate Gaussian that is typically used as the policy distribution in the tasks of physics-based character control. We show that our method performs better than Gaussian baselines in various imitation learning tasks leading to faster learning, higher motion quality and more robustness to external perturbation.

In the experimental section, we showed applications of PFPN to the PPO algorithm. However, our approach is applicable to other common on-policy actor-critic policy gradient DRL algorithms as we show in Appendix C and off-policy methods such as SAC, as we discussed in Section 4.6. Overall, PFPN does not change the underlying architecture or learning mechanism of the DRL algorithms. It is, therefore, complementary to other techniques which improve the policy expressivity given a base action distribution. For example, PFPN can serve as the action policy for each expert in the mixture of experts approach of [Won et al. 2020] or for primitive action learning [Peng et al. 2019]; or as the base distribution of normalizing flows for motion generation [Henter et al. 2020], which we would like to investigate in future work.

As shown in Section 4.5, PFPN is not quite sensitive to the number of particles when more than enough are employed to track the action distribution. However, some fine-tuning may be needed to determine the minimal number of particles necessary to achieve high learning performance with fast convergence speed. In this work, we only considered short-term imitation learning tasks. Thus, further experiments are needed to test the performance of PFPN for tracking long-term motions with heterogeneous behaviors [Bergamin et al. 2019; Won et al. 2020]. Currently, we track each action dimension independently. Accounting for the synergy that exists between different joints has the potential to further improve performance and motion robustness, which opens another exciting avenue for future work.

In any case, we believe that our particle-based action policies provide a great alternative to Gaussian-based actions policies, which have been the staple for DRL-based character control over the past few years. Our work shows significant improvements upon the state-of-the-art in terms of motion quality, robustness to external perturbations, and training efficiency, and we hope that more animation researchers would take advantage of PFPN while training physics-based controllers for continuous control tasks.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant No. IIS-2047632.

REFERENCES

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. 2020. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research* 39, 1 (2020), 3–20.

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics* 38, 6 (2019), 1–11.

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-Based Motion Capture Imitation with Deep Reinforcement Learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. Association for Computing Machinery, Article 1, 10 pages.

Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. 2001. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*. Springer, 3–14.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561* (2018).

Scott Fujimoto, Herke Van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*. 1352–1361.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018a. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018b. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

Perttu Hamalainen, Juuso Toikka, Amin Babadi, and Karen Liu. 2020. Visualizing Movement Control Optimization Landscapes. *IEEE Transactions on Visualization & Computer Graphics* 01 (2020), 1–1.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, Martin Riedmiller, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).

Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. 2020. Moglow: Probabilistic and controllable motion synthesis using normalising flows. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparametrization with Gumble-Softmax. In *International Conference on Learning Representations (ICLR 2017)*.

Wojciech Jaśkowski, Odd Rune Lykkebø, Nihat Engin Toklu, Florian Trifterer, Zdeněk Buk, Jan Koutník, and Faustino Gomez. 2018. Reinforcement Learning to Run… Fast. In *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 155–167.

Yifeng Jiang, Tom Van Wouwe, Friedl De Groote, and C Karen Liu. 2019. Synthesis of biologically realistic human motion using joint torque actuation. *ACM Transactions On Graphics (TOG)* 38, 4 (2019), 1–12.

Augustine Kong, Jun S Liu, and Wing Hung Wong. 1994. Sequential imputations and Bayesian missing data problems. *Journal of the American statistical association* 89, 425 (1994), 278–288.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable muscle-actuated human simulation and control. *ACM Transactions On Graphics (TOG)* 38, 4 (2019), 1–13.

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

Li-Ke Ma, Zeshi Yang, Xin Tong, Baining Guo, and KangKang Yin. 2021. Learning and Exploring Motor Skills with Spacetime Bounds. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 251–263.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. 2017. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201* (2017).

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

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 Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. 2019. MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. *Advances in Neural Information Processing Systems* 32 (2019), 3686–3697.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. 2020. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784* (2020).

Xue Bin Peng and Michiel van de Panne. 2017. Learning locomotion skills using deeprl: Does the choice of action space matter?. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–13.

Avinash Ranganath, Pei Xu, Ioannis Karamouzas, and Victor Zordan. 2019. Low dimensional motor skill learning using coactivation. In *Motion, Interaction and Games*. 1–10.

Daniele Reda, Tianxin Tao, and Michiel van de Panne. 2020. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In *Motion, Interaction and Games*. 1–10.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

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

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

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

Yunhao Tang and Shipra Agrawal. 2018. Boosting trust region policy optimization by normalizing flows policy. *arXiv preprint arXiv:1809.10326* (2018).

Yunhao Tang and Shipra Agrawal. 2019. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500* (2019).

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Transactions On Graphics* 39, 4, Article 33 (2020).

Zhiqi Yin, Zeshi Yang, Michiel Van De Panne, and KangKang Yin. 2021. Discovering diverse athletic jumping strategies. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–17.

A MULTI-MODAL POLICY

In this section, we show the multi-modal representation capacity of PFPN on a one-step bandit task.

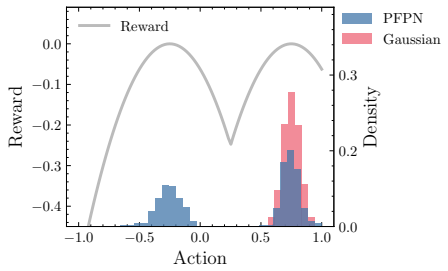


Figure 11: One-step bandit task with asymmetric reward landscape. The reward landscape is defined as the gray line having two peaks asymmetrically at -0.25 and 0.75. The probability densities of stochastic action samples drawn from PFPN (blue) and Gaussian policy (red) are counted after training with a fixed number of iterations.

This is a simple task with one dimension action space $\mathcal{A} = [-1, 1]$. It has an asymmetric 2-peak reward landscape inversely proportional to the minimal distance to points -0.25 and 0.75, as the gray line shown in Figure 11. The goal of this task is to find out the optimal points close to -0.25 and 0.75. In Figure 11, we show the stochastic action sample distributions of PFPN and the naive Gaussian policy after training with the same number of iterations. It is clear that PFPN captures the bi-modal distribution of the reward landscape, while the Gaussian policy gives an unimodal distribution capturing only one of reward peaks.

B HYPERPARAMETERS

Table 2: Default hyperparameters in PFPN baselines.

Parameter	Value
learning rate	$1 \cdot 10^{-4}$
resampling interval	20 environment episodes
dead particle detection threshold (ϵ)	0.0015
discount factor (γ)	0.95
clip range (DPPO)	0.2
GAE discount factor (DPPO, A3C, λ)	0.95
truncation level (IMPALA, $\bar{c}$, $\bar{\rho}$)	1.0
coefficient of policy entropy loss term (A3C, IMPALA)	0.00025
reply buffer size (SAC)	10^6

Since it is infeasible to analytically evaluate the differential entropy of a mixture distribution without approximation, we use the entropy of the categorical distribution for A3C and IMPALA benchmarks, which employ differential entropy during policy optimization.

C ADDITIONAL RESULTS

C.1 Time Complexity

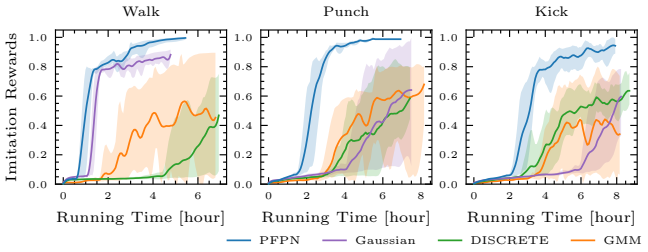


Figure 12: Learning performance as a function of the actual wall clock time using DPPO.

All policies were trained on a machine with Intel 6148G CPU and Nvidia V100 GPU. Training stops when a fixed number of samples is collected as reported in Figure 3. PFPN has a good time consumption performance compared to other baselines. Though the action sampling and particles resampling processes would take extra time, PFPN performs better because its fast convergence avoids wasting time on environment reset when early termination occurs.

C.2 Baselines

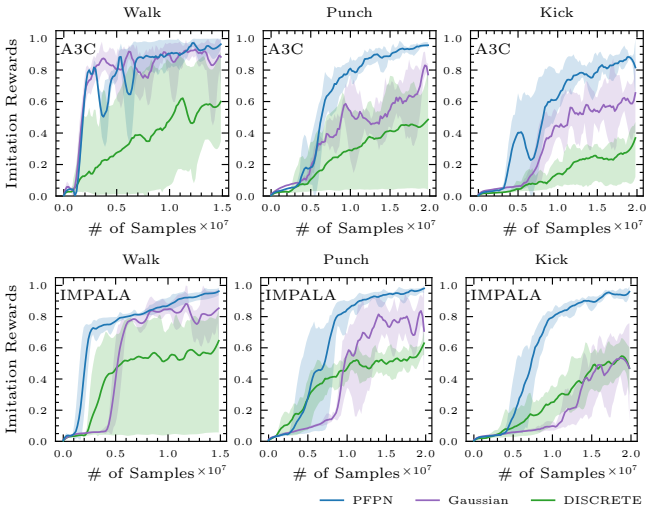


Figure 13: Additional baseline results using A3C and IMPALA.

D POLICY NETWORK LOGITS CORRECTION DURING RESAMPLING

THEOREM. *Let $\mathcal{D}_\tau$ be a set of dead particles sharing the same target particle τ. Let also the logits for the weight of each particle k be generated by a fully-connected layer with parameters $\boldsymbol{\omega}_k$ for the weight and b_k for the bias. The policy $\pi_\theta^{\mathcal{P}}(a_t|\mathbf{s}_t)$ is guaranteed to remain unchanged after resampling via duplicating $\phi_i \leftarrow \phi_\tau, \forall i \in$*

D_τ, if the weight and bias used to generate the unnormalized logits of the target particle are shared with those of the dead one as follows:

$$\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau; \quad b_i, b_\tau \leftarrow b_\tau - \log\left(|\mathcal{D}_\tau| + 1\right). \tag{11}$$

Proof. The weight for the i-th particle is achieved by softmax operation, which is applied to the unnormalized logits L_i, which is the direct output of the policy network:

$$w_i(s_t) = \textsc{softmax}(L_i(s_t)) = \frac{e^{L_i(s_t)}}{\sum_k e^{L_k(s_t)}}. \tag{12}$$

Resampling via duplicating makes dead particles become identical to their target particle. Namely, particles in $\mathcal{D}_\tau \cup \{\tau\}$ will share the same weights as well as the same value of logits, say L'_τ, after resampling. To ensure the policy identical before and after sampling, the following equation must be satisfied

$$\sum_k e^{L_k(s_t)} = \sum_{\mathcal{D}_\tau \cup \{\tau\}} e^{L'_\tau(s_t)} + \sum_{k \notin \mathcal{D}_\tau \cup \{\tau\}} e^{L_k(s_t)} \tag{13}$$

where L_k is the unnormalized logits for the k-th particle such that the weights for all particles who are not in $\mathcal{D}_\tau \cup \{\tau\}$ unchanged, while particles in $\mathcal{D}_\tau \cup \{\tau\}$ share the same weights.

A target particle will not be tagged as dead at all, i.e. $\tau \notin \mathcal{D}_k$ for any dead particle set $\mathcal{D}_k$, since a target particle is drawn according to the particles' weights and since dead particles are defined as the ones having too small or zero weight to be chosen. Hence, Equation 13 can be rewritten as

$$\sum_{i \in \mathcal{D}_\tau} e^{L_i(s_t)} + e^{L_\tau(s_t)} = (|\mathcal{D}_\tau| + 1) e^{L'_\tau(s_t)}, \tag{14}$$

Given that $e^{L_i(s_t)} \approx 0$ for any dead particle $i \in \mathcal{D}_\tau$ and that the number of particles is limited, it implies that

$$e^{L_\tau} \approx (|\mathcal{D}_\tau| + 1) e^{L'_\tau(s_t)}. \tag{15}$$

Taking the logarithm of both sides of the equation leads to that for all particles in $\mathcal{D}_\tau \cup \{\tau\}$, their new logits after resampling should satisfy

$$L'_\tau(s_t) \approx L_\tau(s_t) - \log(|\mathcal{D}_\tau| + 1). \tag{16}$$

Assuming the input of the full-connected layer who generates L_i is $\mathbf{x}(s_t)$, i.e. $L_i(s_t) = \boldsymbol{\omega}_i \mathbf{x}(s_t) + b_i$, we have

$$\boldsymbol{\omega}'_i \mathbf{x}(s_t) + b'_i = \boldsymbol{\omega}_\tau \mathbf{x}(s_t) + b_\tau - \log\left(|\mathcal{D}_\tau| + 1\right). \tag{17}$$

Then, Theorem can be reached. □

If we perform unweighted resampling, it is possible to pick a dead particle as the target particle for some particles. In that case

$$L'_\tau(s_t) \approx L_\tau(s_t) - \log(|D_\tau| + (1 - \sum_k \delta(\tau, \mathcal{D}_k))), \tag{18}$$

where $L'_\tau(s_t)$ is the new logits shared by particles in $\mathcal{D}_\tau$ and $\delta(\tau, \mathcal{D}_k)$ is the Kronecker delta function

$$\delta(\tau, \mathcal{D}_k) = \begin{cases} 1 & \text{if } \tau \in \mathcal{D}_k \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

that satisfies $\sum_k \delta(\tau, \mathcal{D}_k) \leq 1$. Then, for the particle τ, its new logits can be defined as

$$L''_\tau(s_t) \approx (1 - \sum_k \delta(\tau, \mathcal{D}_k)) L'_\tau(s_t) + \sum_k \delta(\tau, \mathcal{D}_k) L_\tau. \tag{20}$$

Consequently, the target particle τ may or may not share the same logits with those in $\mathcal{D}_\tau$, depending on if it is tagged as dead or not.

E VARIANCE OF POLICY GRADIENT IN PFPN CONFIGURATION

Since each action dimension is independent to others, without loss of generality, we here consider the action a_t with only one dimension along which n particles are distributed and the particle i to represent a Gaussian distribution $\mathcal{N}(\mu_i, \sigma_i^2)$. In order to make it easy for analysis, we set up the following assumptions: the reward estimation is constant, i.e. $A_t \equiv A$; logits to support the weights of particles are initialized equally, i.e. $w_i(s_t|\theta) \equiv \frac{1}{n}$ for all particles i and $\nabla_\theta w_1(\mathbf{s}_t|\theta) = \cdots = \nabla_\theta w_n(\mathbf{s}_t|\theta)$; particles are initialized to equally cover the whole action space, i.e. $\mu_i = \frac{i-n}{n}$, $\sigma_i^2 \approx \frac{1}{n^2}$ where $i = 1, \cdots, n$.

From Equation 7, the variance of the policy gradient under such assumptions is

$$\begin{aligned} \mathbb{V}[\nabla_\theta J(\theta)|a_t] &= \int \frac{A_t \sum_i p_i(a_t|\mu_t,\sigma_t) \nabla_\theta w_i(\mathbf{s}_t|\theta)}{\sum_i w_i(s_t|\theta) p_i(a_t|\mu_t,\sigma_t)} a_t^2 \mathrm{d}a_t \\ &\propto \textstyle\sum_i \nabla_\theta w_i(\mathbf{s}_t|\theta) \int a_t^2 p_i(a_t|\mu_t,\sigma_t) \mathrm{d}a_t \\ &\overset{\propto}{\sim} \textstyle\sum_i (\mu_i^2 + \sigma_i^2) \nabla_\theta w_i(\mathbf{s}_t|\theta) \\ &\propto \textstyle\sum_i \frac{(i-n)^2+1}{n^2} \\ &= \frac{n}{3} + \frac{7}{6n} - \frac{1}{2} \\ &\sim 1 - \frac{3}{2n} + O(\frac{1}{n^2}). \end{aligned} \tag{21}$$

Given $\mathbb{V}[\nabla_\theta J(\theta)|a_t] = 0$ when $n = 1$, from Equation 21, for any $n > 0$, the variance of policy gradient $\mathbb{V}[\nabla J(\theta)|a_t]$ will increase with n. Though the assumptions usually are hard to meet perfectly in practice, this still gives us an insight that employing a large number of particles may result in more challenge to optimization.

This conclusion is consistent with that in the case of uniform discretization [Tang and Agrawal 2019] where the variance of policy gradient is shown to satisfy

$$\mathbb{V}[\nabla_\theta J(\theta)|a_t]_{\text{DISCRETE}} \sim 1 - \frac{1}{n}. \tag{22}$$

That is to say, in either PFPN or uniform discretization scheme, we cannot simply improve the control performance of the police by employing more atomic actions, i.e. by increasing the number of particles or using more bins in the uniform discretization scheme, since the gradient variance increases as the discretization resolution increases. However, PFPN has a slower increase rate, which implies that it might support more atomic actions before performance drops due to the difficulty in optimization. Additionally, compared to the fixed, uniform discretization scheme, atomic actions represented by particles in PFPN are movable and their distribution can be optimized. This means that PFPN has the potential to provide better discretization scheme using fewer atomic actions to meet the fine control demand and thus be more friendly to optimization using policy gradient.

F PFPN WITH OFF-POLICY POLICY GRADIENT ALGORITHMS

To enable PFPN applicable in state-action value based off-policy algorithms, we propose a reparamterization trick in this section

such that a sampled action $a_\theta^{\mathcal{P}}(\mathbf{s}_t)$ can be differentiable to the policy network parameter θ.

F.1 Reparameterization Trick

Let $\mathbf{x}(\mathbf{s}_t|\theta) \sim \textsc{Concrete}(\{w_i(\mathbf{s}_t|\theta); i = 1, 2, \cdots\}, \lambda)$ is a sampling result of a relaxed version of the one-hot categorical distribution supported by the probability of $\{w_i(\mathbf{s}_t|\theta); i = 1, 2, \cdots\}$, where $\mathbf{x}(\mathbf{s}_t|\theta) = \{x_i(\mathbf{s}_t|\theta); i = 1, 2, \cdots\}$ is reparametrizable and λ is picked to be 1 in our implementation. We apply the Gumbel-softmax trick [Jang et al. 2017] to get a sampled action value as

$$a'(\mathbf{s}_t) = \textsc{stop}\left(\sum_i a_i \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta))\right), \tag{23}$$

where a_i is the sample drawn from the distribution represented by the particle i with parameter ϕ_i, $\textsc{stop}(\cdot)$ is a "gradient stop" operation, and $\delta(\cdot,\cdot)$ denotes the Kronecker delta function. Then, the reparameterized sampling result can be written as follows:

$$a_\theta^{\mathcal{P}}(\mathbf{s}_t) = \sum_i (a_i - a'(\mathbf{s}_t)) m_i + a'(\mathbf{s}_t)\delta(i, \arg\max \mathbf{x}) \equiv a'(\mathbf{s}_t), \tag{24}$$

where $m_i := x_i(\mathbf{s}_t|\theta) + \textsc{stop}(\delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta)) - x_i(\mathbf{s}_t|\theta)) \equiv \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta))$ composing a one-hot vector that approximates the samples drawn from the corresponding categorical distribution. Since $x_i(\mathbf{s}_t|\theta)$ drawn from the concrete distribution is differentiable to the parameter θ, the gradient of the reparameterized action sample can be obtained by

$$\begin{aligned} &\nabla_\theta a_\theta^{\mathcal{P}}(\mathbf{s}_t) = \textstyle\sum_i (a_i - a'(\mathbf{s}_t)) \nabla_\theta x_i(\mathbf{s}_t|\theta); \\ &\nabla_{\phi_i} a_\theta^{\mathcal{P}} = \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta)) \nabla_{\phi_i} a_i. \end{aligned} \tag{25}$$

Through these equations, both the policy network parameter θ and the particle parameters ϕ_i can be updated by backpropagation through the sampled action $a'(\mathbf{s}_t)$.

F.2 Policy Representation with Action Bounds

In off-policy algorithms, like DDPG and SAC, an invertible squashing function, typically the hyperbolic tangent function, will be applied to enforce action bounds on samples drawn from Gaussian distributions, e.g. in SAC, the action for the k-th dimension at the time step t is obtained by

$$a_{t,k}(\varepsilon, \mathbf{s}_t) = \tanh u_{t,k} \tag{26}$$

where $u_{t,k} \sim \mathcal{N}(\mu_\theta(\mathbf{s}_t), \sigma_\theta^2(\mathbf{s}_t))$, $\mu_\theta(\mathbf{s}_t)$ and $\sigma_\theta^2(\mathbf{s}_t)$ are parameters generated by the policy network with parameter θ, and $u_{t,k}$ can be written $u_{t,k} = \mu_\theta(\mathbf{s}_t) + \xi_{t,k}\sigma_\theta^2(\mathbf{s}_t)$ given a noise variable $\xi_{t,k} \sim \mathcal{N}(0, 1)$ such that $a_{t,k}$ is reparameterizable.

Let $\mathbf{a}_t = \{\tanh u_{t,k}\}$ where $u_{t,k}$, drawn from the distribution represented by a particle with parameter $\phi_{t,k}$, is a random variable sampled to support the action on the k-th dimension. Then, the probability density function of PFPN represented by Equation 3 can be rewritten as

$$\pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \prod_k \sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\phi_{i,k}) / (1 - \tanh^2 u_{t,k}), \tag{27}$$

and the log-probability function becomes

$$\begin{aligned} \log \pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) \quad &= \textstyle\sum_k \log \Big[\sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\phi_{i,k}) \\ &\quad -2\left(\log 2 - u_{t,k} - \text{softplus}(-2u_{t,k})\right)\Big]. \end{aligned} \tag{28}$$

In our SAC implementation, we use Gaussian noises to generate action samples, i.e. $u_{i,k} \sim \mathcal{N}(\mu_{i,k}, \xi_{i,k}^2)$ where $\mu_{i,k}$ and $\xi_{i,k}$ are the parameters for the i-th particle at the k-th action dimension.