# A Algorithm

**Algorithm 1:** Policy Gradient Method using PFPN

Initialize the neural network parameter $\theta$ and learning rate $\alpha$;
initialize particle parameters $\phi_i$ to uniformly distribute particles on each action dimension;
initialize the threshold $\epsilon$ 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**
        $\tau_i \sim P\left(\cdot|\mathbb{E}\left[w_k|w_k \in \mathcal{W}_k\right], k = 1, 2, \cdots\right)$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_i\}$; $\mathcal{D}_{\tau_i} \leftarrow \mathcal{D}_{\tau_i} \cup \{i\}$
      **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.*
        $\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_\tau$; $b_i \leftarrow b_\tau - \log(|\mathcal{D}_\tau| + 1)$
      **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**

# B Policy Network Logits Correction during Resampling

**Theorem.** *Let $\mathcal{D}_\tau$ be a set of dead particles sharing the same target particle $\tau$. 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 $\phi_i \leftarrow \phi_\tau, \forall i \in D_\tau$, 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:*

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

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

# C PFPN with Off-Policy Policy Gradient Algorithms

To enable PFPN applicable in state-action value based off-policy algorithms, we propose a reparameterization trick in this section such that a sampled action $a_\theta^{\mathcal{P}}(\mathbf{s}_t)$ can be differentiable to the policy network parameter $\theta$.

## C.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 $\lambda$ 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

$$a'(\mathbf{s}_t) = \textsc{stop}\left(\sum_i a_i \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta))\right), \tag{21}$$

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

$$a_\theta^{\mathcal{P}}(\mathbf{s}_t) = \sum_i (a_i - a'(\mathbf{s}_t))m_i + a'(\mathbf{s}_t)\delta(i, \arg\max \mathbf{x}) \equiv a'(\mathbf{s}_t), \tag{22}$$

where $m_i := x_i(\mathbf{s}_t|\theta) + \text{STOP}(\delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta)) - x_i(\mathbf{s}_t|\theta)) \equiv \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta))$ 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 $\theta$, the gradient of the reparameterized action sample can be obtained by

$$\begin{aligned} \nabla_\theta a_\theta^{\mathcal{P}}(\mathbf{s}_t) &= \textstyle\sum_i (a_i - a'(\mathbf{s}_t))\nabla_\theta x_i(\mathbf{s}_t|\theta); \\ \nabla_{\phi_i} a_\theta^{\mathcal{P}} &= \delta(i, \arg\max \mathbf{x}(\mathbf{s}_t|\theta))\nabla_{\phi_i} a_i. \end{aligned} \tag{23}$$

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

### C.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

$$a_{t,k}(\varepsilon, \mathbf{s}_t) = \tanh u_{t,k} \tag{24}$$

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 $\theta$, 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. 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.

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

$$\pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \prod_k \sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\phi_{i,k}) / (1 - \tanh^2 u_{t,k}), \tag{25}$$

and the log-probability function becomes

$$\log \pi_\theta^{\mathcal{P}}(\mathbf{a}_t|\mathbf{s}_t) = \sum_k \log \left[ \sum_i w_{i,k}(\mathbf{s}_t|\theta) p_{i,k}(u_{t,k}|\phi_{i,k}) - 2\left(\log 2 - u_{t,k} - \text{softplus}(-2u_{t,k})\right) \right]. \tag{26}$$

## D Hyperparameters

Table 2: Default hyperparameters in PFPN baselines.

| Parameter | Value |
|---|---|
| learning rate | $1 \cdot 10^{-4}$ |
| resampling interval | 1200 steps |
| dead particle detection threshold ($\epsilon$) | 0.0015 |
| discount factor ($\gamma$) | 0.95 |
| clip range (DPPO) | 0.2 |
| GAE discount factor (DPPO, A3C, $\lambda$) | 0.95 |
| truncation level (IMPALA, $\bar{c}$, $\bar{\rho}$) | 1.0 |
| entropy loss coefficient (A3C, IMPALA) | 0.00025 |
| reply buffer size (SAC) | $10^6$ |

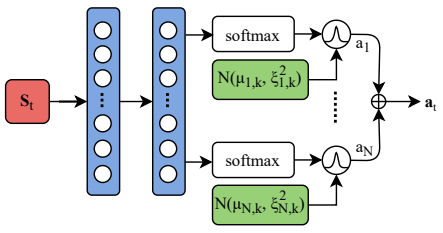



Figure 9: PFPN architecture with a $N$-dimension action space in our experiment. $\oplus$ denotes the concatenation operator.

We set the resampling interval as 1200 environment steps, which are 20 environment episodes of simulation. 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. In our 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 9. 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.

# E Additional Results

## E.1 Time Complexity

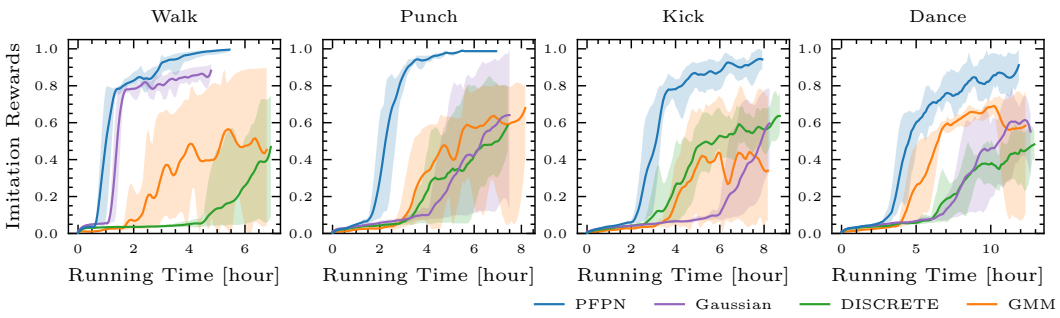



Figure 10: Learning performance as a function of the actual wall clock time using DPPO.

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 2. 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.

## E.2 Baselines

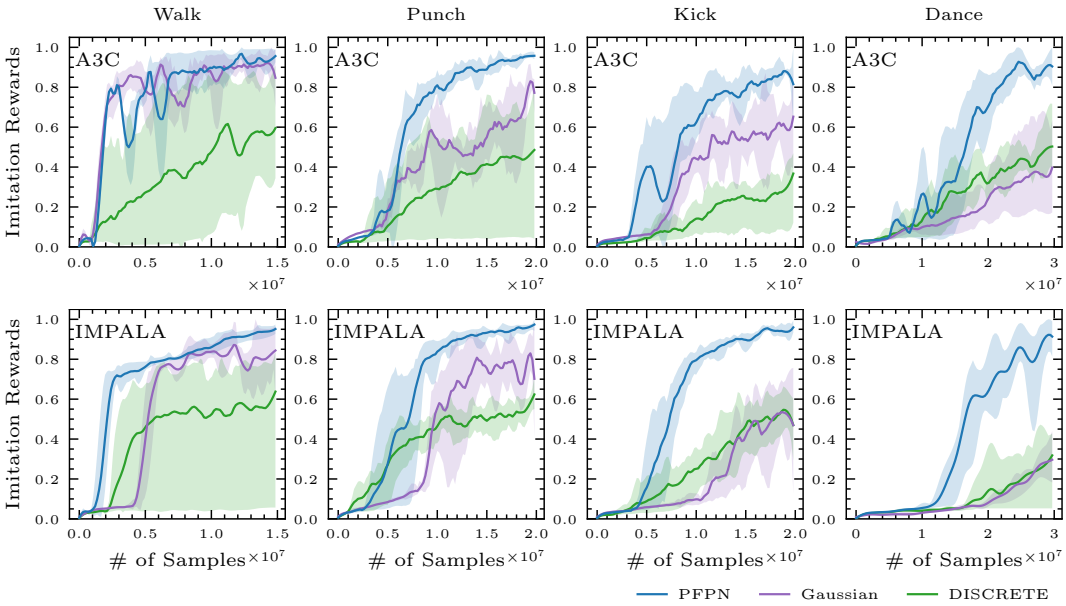



Figure 11: Additional baseline results using A3C and IMPALA.

# F 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

$$\begin{aligned}
\mathbb{V}[\nabla_\theta J(\theta)|a_t] &= \int \frac{A_t \sum_i p_i(a_t|\mu_t,\sigma_t)\nabla_\theta w_i(\mathbf{s}_t|\theta)}{\sum_i w_i(s_t|\theta)p_i(a_t|\mu_t,\sigma_t)} a_t^2 \mathrm{d}a_t \\
&\propto \textstyle\sum_i \nabla_\theta w_i(\mathbf{s}_t|\theta) \int a_t^2 p_i(a_t|\mu_t,\sigma_t)\mathrm{d}a_t \\
&\overset{\propto}{\sim} \textstyle\sum_i (\mu_i^2 + \sigma_i^2)\nabla_\theta w_i(\mathbf{s}_t|\theta) \\
&\propto \textstyle\sum_i \frac{(i-n)^2+1}{n^2} \\
&= \frac{n}{3} + \frac{7}{6n} - \frac{1}{2} \\
&\sim 1 - \frac{3}{2n} + O(\frac{1}{n^2}).
\end{aligned} \tag{27}$$

Given $\mathbb{V}[\nabla_\theta J(\theta)|a_t] = 0$ when $n = 1$, from Equation 27, 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

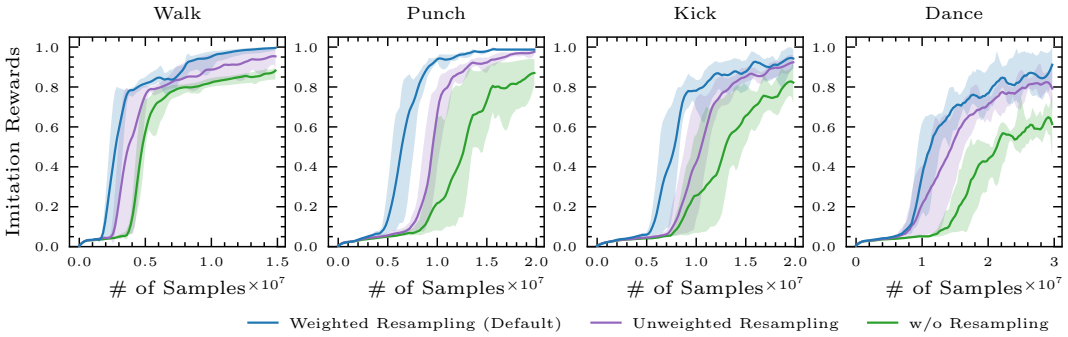
$$\mathbb{V}[\nabla_\theta J(\theta)|a_t]_{\text{DISCRETE}} \sim 1 - \frac{1}{n}. \tag{28}$$

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.

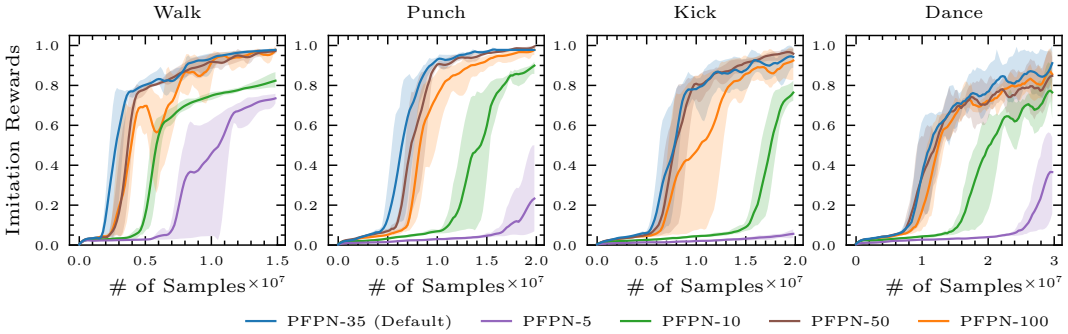
# G Ablation Study

**Resampling Strategy.** We consider two resampling strategies as comparison: (1) the weighted resampling strategy described in Section 3.3 that draws targets for dead particles according to the weights of the remaining, alive ones, and (2) the unweighted resampling strategy that picks a $\tau_i$ randomly from all alive particles. In Figure 12a, we also perform comparison to PFPN without any resampling. The weighted resampling strategy 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.

**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




(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 12: Sensitivity of PFPN to different resampling strategies and the number of particles.

fine control more possible. However, in Appendix F, 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 12b. 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.