Engineering Applications of Artificial Intelligence 123 (2023) 106465

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Distributional and hierarchical reinforcement learning for physical systems with noisy state observations and exogenous perturbations

Jehyun Park [a], Jongeun Choi [a,b,*], Sungjae Nah [a], Dohee Kim [c]

[a] School of Mechanical Engineering, Yonsei University, Seoul, 03722, Republic of Korea
[b] Department of Artificial Intelligence, Yonsei University, Seoul, 03722, Republic of Korea
[c] Electrified Systems Control Research Laboratory, Research and Development Division, Hyundai Motor Company, Hwaseong, Gyeonggi-do, 18280, Republic of Korea

ARTICLE INFO

Keywords:
Hierarchical reinforcement learning
Distributional reinforcement learning
Noisy state observation
Exogenous perturbation

ABSTRACT

Reinforcement learning has shown remarkable success in various applications, and in some cases, even outperforms human performance. However, despite the potential of reinforcement learning, numerous challenges still exist. In this paper, we introduce a novel approach that exploits the synergies between hierarchical reinforcement learning and distributional reinforcement learning to address complex sparse-reward tasks, where noisy state observations or non-stationary exogenous perturbations are present. Our proposed method has a hierarchical policy structure, where random rewards are modeled as random variables that follow a value distribution. This approach enables the handling of complex tasks and increases robustness to uncertainties arising from measurement noise or exogenous perturbations, such as wind. To achieve this, we extend the distributional soft Bellman operator and temporal difference error to include the hierarchical structure, and we use quantile regression to approximate the reward distribution. We evaluate our method using a bipedal robot in the OpenAI Gym environment and an electric autonomous vehicle in the SUMO traffic simulator. The results demonstrate the effectiveness of our approach in solving complex tasks with the aforementioned uncertainties when compared to state-of-the-art methods. Our approach demonstrates promising results in handling uncertainties caused by noise and perturbations for challenging sparse-reward tasks, and could potentially pave the way for the development of more robust and effective reinforcement learning algorithms in real physical systems.

1. Introduction

Recently, reinforcement learning (RL) has made substantial advances and gained popularity with ground-breaking publications (Mnih et al., 2013, 2015). As a result, it is becoming more widely used in various fields (Han et al., 2022; Ha et al., 2021; Smith et al., 2022; Kim et al., 2021; Ghadirzadeh et al., 2020; Zheng et al., 2022; Ji et al., 2022; Yong et al., 2022; Pylorof and Garcia, 2022; Cho et al., 2022; Samsonov et al., 2022; Lim et al., 2020; Kim et al., 2020). Furthermore, RL achieves superhuman performance even in complex and challenging domains that require accurate and nuanced predictions over large search spaces, such as the game of Go (Silver et al., 2017) and racing (Wurman et al., 2022).

Unlike supervised learning (Mohri et al., 2018), which involves training a model on labeled data to teach the learner to distinguish between good and bad behaviors, RL relies on the agent learning to optimize its actions through interactions with the environment to maximize cumulative rewards. RL algorithms have been developed in various ways to address different needs (Van Hasselt et al., 2016; Mnih et al., 2016; Lillicrap et al., 2015; Haarnoja et al., 2018; Levine et al., 2020). Soft Actor–Critic (SAC) (Haarnoja et al., 2018) is one of the RL algorithms that combines an expected reward with an entropy term that encourages exploration. SAC has been shown to achieve state-of-the-art performance on a wide range of control tasks. However, despite the progress in the development of RL algorithms, there are still challenges that need to be addressed.

Due to the nature of the learning structure, RL methods may encounter difficulties in achieving the sample efficiency in collecting data, scalability to a large state space, generalization to unseen environments, and robustness to perturbations (Flet-Berliac, 2019; Sun et al., 2021). One of challenging tasks to solve with RL algorithms is the sparse reward task. In these tasks, agents are only rewarded when they are close enough to their goals.

Next, noisy state observations and exogenous perturbations are two types of factors that can significantly impact the performance of RL algorithms. Noisy state observations occur when an agent's observations of the environment are subject to random or stochastic noise, which can

* Corresponding author at: School of Mechanical Engineering, Yonsei University, Seoul, 03722, Republic of Korea.
E-mail addresses: jhyunpark@yonsei.ac.kr (J. Park), jongeunchoi@yonsei.ac.kr (J. Choi), sjnah@yonsei.ac.kr (S. Nah), doheekim@hyundai.com (D. Kim).

https://doi.org/10.1016/j.engappai.2023.106465
Received 30 November 2022; Received in revised form 21 April 2023; Accepted 11 May 2023
Available online 25 May 2023
0952-1976/© 2023 Elsevier Ltd. All rights reserved.

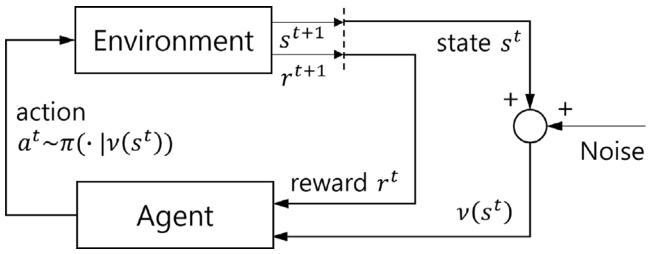


Fig. 1. Reinforcement learning mechanism with added noise to the state observations. The exogenous perturbations are contained in the environment.

make it more difficult for the agent to accurately estimate the true state of the environment. Exogenous perturbations, on the other hand, refer to external disturbances or changes to the environment that are beyond the control of the agent. These perturbations can include changes to the state of the environment, resulting in changes to the reward values, which can make it more difficult for the agent to learn a robust policy. In both cases, the quality of the learned policy may be impacted, and RL algorithms may need to be adapted or improved to account for these factors.

We aim to address the challenges of sample efficiency and robustness to perturbations in RL. Sparse rewards for goal achievement result in low sample efficiency, while measurement noise and exogenous perturbations such as wind undermine robustness. Although existing literature deals with each challenge separately, we aim to address them together. We first briefly review existing approaches before introducing our novel method. To solve the challenging sparse-reward tasks, hierarchical RL (HRL) (Gehring et al., 2021) decomposes tasks into hierarchies of subtasks to improve the efficiency of exploration (Lee and Choi, 2022; Pateria et al., 2021; Li and Ding, 2023). HIerarchical Reinforcement learning with Off-policy correction (HIRO) (Nachum et al., 2018) proposes to use off-policy experience for both higher- and lower-level training to achieve high sample efficiency. Switching Ensemble (SE) (Nachum et al., 2019) presents exploration techniques inspired by the hierarchy that achieve performance competitive with HRL methods. However, it is difficult to apply general RL algorithms, including HRL, to physical real-world systems in practice due to measurement noise in the state observations (see Fig. 1), which can be viewed as a special case of a partially observable environment. In addition, in environments where rewards and state transitions are inherently stochastic, traditional RL algorithms that predict the average over all potential rewards have limited performance (Lowet et al., 2020). To deal with this issue, distributional RL (DRL) (Bellemare et al., 2017; Mavrin et al., 2019) may be used to model future rewards as a distribution called a value distribution to capture uncertainties of the environment, in contrast to the standard RL approach. The key to this approach is that the overall distribution contains far more information than the expectation by itself, e.g., multimodal distributions, which can be used to guide behaviors that ultimately lead to superior rewards.

In this paper, our goal is to develop a practical method for solving complex sparse-reward tasks in situations where the measurements are uncertain. We propose to simultaneously exploit HRL and DRL to achieve our goal. Compared to HRL (when DRL is ablated), our method shows better performance against uncertainties by modeling random rewards as random variables via a value distribution. Compared to DRL (when HRL is ablated), our method has a hierarchical structure of policies that can deal with more complex tasks. We integrate the two methods in a novel way to satisfy convergence properties outlined in the lemmas in Section 3. In particular, we use SAC as the base RL algorithm for learning robust policies by encouraging efficient exploration. To tackle a complex sparse-reward task, our approach builds on the hierarchical structure from Gehring et al. (2021). It consists of a three-level hierarchical policy: the highest-level policy selects which joints the robot will use, the second-level policy determines the target robot configuration, and the pre-trained low-level policy controls the robot to reach the target configuration. These policies, except the low-level policy, are updated based on the value distribution. Therefore, it is crucial to approximate the value distribution, and we achieve this by using quantile regression. In addition, since our agent has a hierarchical structure, we show how to modify the distributional soft Bellman operator and the temporal difference error to deal with multiple actions from the policies.

Our contributions are as follows. First, we propose a novel method called distributional and hierarchical RL (DHRL), which takes advantage of HRL and DRL to address the aforementioned challenging issues. We show how to seamlessly integrate both methods in a synergetic way to deal with such challenges using the proposed DHRL algorithm. Next, we provide convergence properties for the proposed DHRL algorithm. We then evaluate our method using various benchmark tasks such as (a) Hurdles, (b) Limbo, and (c) Gaps implemented with the MuJoCo physics simulator (Todorov et al., 2012) in OpenAI Gym (Brockman et al., 2016). We also apply DHRL to an autonomous electric vehicle navigating urban streets and show that it outperforms HRL in terms of ride comfort and battery consumption. Our method is shown to be effective when there are uncertainties in observations and/or non-stationary exogenous perturbations during the training and testing processes, which is a practical situation. In this setting, our method shows robust performance and is expected to be useful in many real-world robotics applications with noisy sensory measurements or exogenous perturbations. Finally, we have successfully demonstrated the effectiveness of our method via ablation studies that solve complex tasks in situations where the measurements are uncertain. Our method outperforms each of the state-of-the-art RL algorithms alone.

The rest of our paper is organized as follows. Section 2 introduces the background the development of our method. Section 3 provides an exposition of our method. Sections 4 and 5 present simulation results that show the efficacy of our method, and Section 6 concludes our study.

2. Background

2.1. Reinforcement learning

RL is a type of machine learning where an agent learns to make decisions by interacting with an environment and receiving rewards or penalties for its actions. The agent's goal is to learn a policy that maximizes its cumulative reward over time. To do this, the agent interacts with the environment by observing its current state and choosing actions based on its policy. The agent then receives a reward signal from the environment, which it uses to update its policy. This process continues iteratively, with the agent continuously adjusting its policy based on the rewards it receives. Eventually, the agent's policy should converge to an optimal policy that maximizes the cumulative reward. RL algorithms use a variety of techniques such as Markov Decision Process (MDP), Q-Learning, and policy gradients to represent and update policies.

Let us consider an MDP defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ denote a state space and an action space, respectively. $\mathcal{R}$ denotes a reward function and $\mathcal{P}(s'|s,a)$ represents a transition probability from a state s to a state s' after taking an action a. γ denotes a discount factor. Let policy π be a probability distribution that maps from a state s to an action a. RL updates the policy by maximizing the action-value function Q. Furthermore, SAC adds an entropy term to the action-value function to learn a policy by maximizing the standard reward function and the maximum entropy objective together (Ma et al., 2020),

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t [\mathcal{R}(s_t, a_t) - \alpha \log \pi(a_{t+1}|s_{t+1})]\right],$$

where α is a temperature parameter that balances the entropy and the reward.

The policy is updated through a policy iteration, which alternates policy evaluation and policy improvement steps. The policy evaluation step is performed by repeatedly applying the Bellman operator, i.e., $Q^{k+1} = \mathcal{T}^{\pi} Q^{k}$, the sequence of Q^{k} converges to the value of a fixed policy, Q^{π}, as $k \to \infty$ (Haarnoja et al., 2018). During the policy improvement step, the policy is improved to have a higher Q-value by solving the minimization problem using the Kullback–Leibler divergence (Haarnoja et al., 2018):

$$\pi_{new} = \arg\min_{\pi' \in \Pi} D_{KL}\left(\pi'(\cdot|s_t) \parallel \frac{\exp(\frac{1}{\alpha}Q^{\pi_{old}}(s_t,\cdot))}{\Delta^{\pi_{old}}(s_t)}\right), \tag{1}$$

where $\Delta^{\pi_{old}}(s_t)$ is the partition function that normalizes the distribution.

2.2. Hierarchical reinforcement learning

Since our approach builds on the hierarchical structure from Gehring et al. (2021), we first introduce their approach in this section. As one of the HRL algorithms, Gehring et al. (2021) decomposes a given task into low-level skills and high-level control signals, allowing the algorithm to trade-off between the sample efficiency and generality. In particular, the hierarchical policy has a three-level hierarchy that enables trade-offs between general and specific skills required for each task. Task-specific information is provided only for the two high-level policies to guide exploration, and the low-level policy is pre-trained in an unsupervised manner to achieve generality. The high-level policy π^{d} selects a combination of features $a_d \in \mathcal{A}^{d}$, the second-level policy π^{c} determines the target configuration $a_c \in \mathcal{A}^{c}$ within the selected combination, and the pre-trained low-level policy π^{lo} moves the agent to reach the target configuration. Gehring et al. (2021) extends SAC with a shared critic $Q(s, a_d, a_c)$ to jointly learn π^{d} and π^{c}. The value function $V(s)$ is modified as follows (Gehring et al., 2021):

$$V(s) = \sum_{a_d \in \mathcal{A}^d} \pi^{d}(a_d|s) \mathop{\mathbb{E}}_{a_c\sim\pi^c}\left[Q(s,a_d,a_c) - \frac{\beta}{|a_d|}\log \pi^{c}(a_c|s,a_d)\right] + \alpha(\mathcal{H}(\pi^{d}(\cdot|s)) - \log|\mathcal{A}_d|),$$

where $s \in \mathcal{S}$ is a state observation and $\mathcal{H}$ denotes an entropy. α and β are temperatures that determine the relative importance of the entropy of π^{d} and π^{c}, respectively. β is normalized with $|a_d|$ to take into account different action dimensions across the target configurations. The policy loss is given as (Gehring et al., 2021)

$$\mathop{\mathbb{E}}_{s\sim B}\left[\sum_{a_d \in \mathcal{A}^d} \pi^{d}(a_d|s) \mathop{\mathbb{E}}_{a_c\sim\pi^c}\left[\frac{\beta}{|a_d|}\log \pi^{c}(a_c|s,a_d) - Q(s,a_d,a_c)\right] - \alpha\mathcal{H}(\pi^{d}(\cdot|s))\right],$$

where B is a replay buffer.

2.3. Distributional reinforcement learning

In contrast to standard RL, DRL considers the distribution of the reward instead of the expectation of the reward. By exploiting the distributional information, DRL methods can achieve better performance (Ma et al., 2020). We first introduce their approach in this section. In DRL, the distributional Bellman operator for DRL is defined as follows (Ma et al., 2020):

$$\mathcal{T}^{\pi} Z(s,a) \overset{D}{:=} \mathcal{R}(s,a) + \gamma Z(s',a'), \tag{2}$$

where $U \overset{D}{:=} V$ indicates that random variable U is distributed as same as V, and Z is the value distribution whose expectation is Q as in the standard RL formulation (Ma et al., 2020).

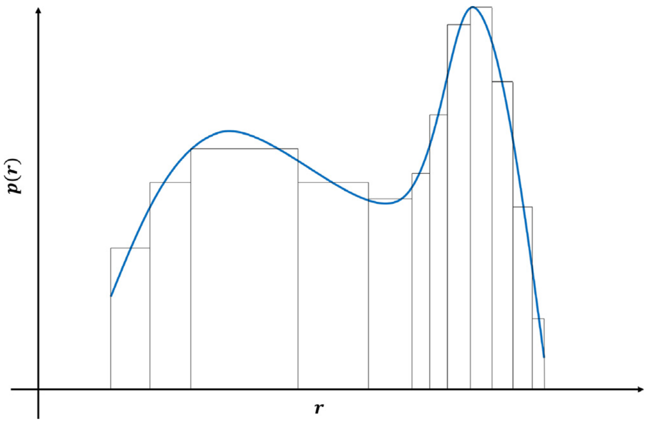


Fig. 2. An example of approximating a value distribution for a given state and action. The X-axis and Y-axis denote a reward and the probability of receiving that reward, respectively.

In order to approximate the value distribution Z, we introduce the Wasserstein metric d_p between two distributions U and V as (Bellemare et al., 2017)

$$d_p(U,V) = \left(\int_0^1 |F_U^{-1}(\omega) - F_V^{-1}(\omega)|^p d\omega\right)^{1/p},$$

where F_U and F_V are cumulative distribution functions (CDF) of U and V for $p < \infty$. Let $\mathcal{Z}$ denotes the value distribution space with bounded moments. A maximal form of the Wasserstein metric for two value distributions $Z_1, Z_2 \in \mathcal{Z}$ is defined as (Bellemare et al., 2017)

$$\bar{d}_p(Z_1,Z_2) := \sup_{s,a} d_p(Z_1(s,a), Z_2(s,a)).$$

The distributional Bellman operator (2) is a γ-contraction in $\bar{d}_p$, the sequence $Z^{k+1} = \mathcal{T}^{\pi} Z^{k}$ converges to Z^{π} (Bellemare et al., 2017).

Similar to SAC, considering the entropy of the policy, the value distribution is defined as (Ma et al., 2020)

$$Z^{\pi}(s,a) \overset{D}{:=} \sum_{t=0}^{\infty} \gamma^{t}\left[\mathcal{R}(s_t,a_t) - \alpha \log \pi(a_{t+1}|s_{t+1})\right], \quad s_{t+1} \sim \mathcal{P}(\cdot|s_t,a_t), a_t \sim \pi(\cdot|s_t).$$

The distributional Bellman operator (2) is then modified as (Ma et al., 2020)

$$\mathcal{T}^{\pi} Z(s,a) \overset{D}{:=} \mathcal{R}(s,a) + \gamma\left[Z(s',a') - \alpha\log\pi(a'|s')\right], \quad s' \sim \mathcal{P}(\cdot|s,a), a' \sim \pi(\cdot|s'). \tag{3}$$

This operator keeps convergence properties from the original operator (2).

2.4. Approximating value distribution

Since the DRL policy is updated based on the value distribution, estimating the value distribution is the main challenge. Fig. 2 shows an example of how to approximate an arbitrary distribution with a finite number of sample points. The number of sample points and the spacing between them affect how close the approximation is to the true distribution. There are several ways to select sample points, which will be discussed later in this section.

The value distribution, a type of probability distribution, indicates how probabilities are distributed over the rewards. In order to approximate the value distribution, DRL approaches (Dabney et al., 2018b; Bellemare et al., 2017; Ma et al., 2020) represent it using the quantile function F_Z^{-1} (Müller, 1997), which is the inverse function of the CDF ($F_Z(z) = Pr(Z < z)$). By definition, we have $F_Z^{-1}(\tau) := \inf\{z \in \mathbb{R} : \tau \leq F_Z(z)\}$. In our scenario, z is a random reward that the

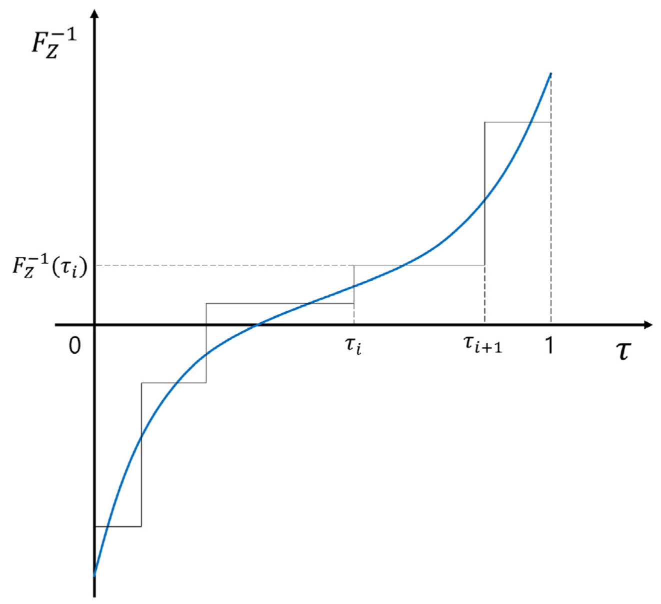


Fig. 3. An example of approximating a quantile function with a finite number of points.

agent can receive. Given a state and an action, z can be any value according to its probability distribution. Therefore, if we know the distribution or the quantile function $F_Z^{-1}(\tau)$, we can predict how much reward the agent will receive. However, since it is hard to recover the unknown distribution exactly, we approximate the distribution with a finite number of points (see Fig. 3). Each point is defined with a quantile value $F_Z^{-1}(\tau_i)$ at a quantile fraction τ_i. Quantile regression is used to find quantile values at given quantile fractions, and uses data stored in the replay buffer.

Previous studies have proposed several methods for generating quantile fractions: uniformly distributed (Dabney et al., 2018b), randomly sampled (Dabney et al., 2018a), and generated using a fraction proposal network (Yang et al., 2019). In this paper, we follow the method from Yang et al. (2019) to approximate the value distribution. According to Yang et al. (2019), we first generate the set of quantile fractions using the fraction proposal network FP and then compute the corresponding quantile values using the quantile value network $F_{Z,\psi}^{-1}$. Since the true quantile function F_Z^{-1} is unknown in practice, we use the quantile value network $F_{Z,\psi}^{-1}$ parameterized with ψ instead. Then the value distribution Z is approximated as (Yang et al., 2019)

$$Z_{\theta,\tau}(s,a) := \sum_{i=0}^{N-1}(\tau_{i+1}-\tau_i)\delta_{\theta_i(s,a)},$$

where $\{\theta_i\}$ and $\{\tau_i\}$ denote quantile values and quantile fractions, respectively. δ_z denotes a Dirac at $z \in \mathbb{R}$, and $\{\tau_i\}_{i=0,\ldots,N}$ satisfy $\tau_{i-1} < \tau_i$ with $\tau_0 = 0$ and $\tau_1 = 1$. Given a set of quantile fractions from the fraction proposal network, the temporal difference error between two quantile fractions $\hat{\tau}_i$ and $\hat{\tau}_j$ is given by (Ma et al., 2020)

$$\delta_{ij}^t = r_t + \gamma\left[F_{Z',\psi}^{-1}(\hat{\tau}_i) - \alpha\log\pi(a'|s')\right] - F_{Z,\psi}^{-1}(\hat{\tau}_j), \tag{4}$$

where $\hat{\tau}_i$ denotes $\frac{\tau_i+\tau_{i+1}}{2}$. For training the quantile value network, we use quantile regression, a popular method for approximating the quantile functions of distributions by minimizing the Huber quantile regression loss (Huber, 1992) with threshold κ:

$$\rho_\tau^\kappa(\delta_{ij}) = \left|\tau - \mathbb{I}\{\delta_{ij}<0\}\right|\frac{\mathcal{L}_\kappa(\delta_{ij})}{\kappa}, \text{with}$$

$$\mathcal{L}_\kappa(\delta_{ij}) = \begin{cases}\frac{1}{2}\delta_{ij}^2 & \text{if } |\delta_{ij}| \le \kappa \\ \kappa\left(|\delta_{ij}| - \frac{1}{2}\kappa\right), & \text{otherwise}\end{cases}$$

The quantile value network is then trained by minimizing the following objective (Yang et al., 2019):

$$\mathcal{L}(s_t,a_t,r_t,s_{t+1}) = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\rho_{\hat{\tau}_j}^\kappa(\delta_{ij})$$

With the fixed quantile value network, the fraction proposal network is trained by minimizing 1-Wasserstein metric W_1 (Yang et al., 2019),

$$W_1(Z,\tau) = \sum_{i=0}^{N-1}\int_{\tau_i}^{\tau_{i+1}}\left|F_{Z,\psi}^{-1}(\omega) - F_{Z,\psi}^{-1}(\hat{\tau}_i)\right| d\omega.$$

The above minimization problem is solved by repeatedly applying gradient descent as follows (Yang et al., 2019):

$$\frac{\partial W_1}{\partial\tau_i} = 2F_{Z,\psi}^{-1}(\tau_i) - F_{Z,\psi}^{-1}(\hat{\tau}_i) - F_{Z,\psi}^{-1}(\hat{\tau}_{i-1}).$$

3. Method

In this paper, our approach builds on the hierarchical structure from Gehring et al. (2021) but updates the policies based on a DRL approach. The overall procedure of our method is illustrated in Fig. 4. Our method consists of two large loops: one that approximates the value distribution (left-hand side of Fig. 4), and the other, the RL loop (right-hand side of Fig. 4). Basically, the RL agent interacts with the environment in the RL loop. While the RL agent performs an action based on its policies, data samples are stored in the replay buffer. At a certain interval, the policies are updated based on value distributions for different pairs of states and actions. In addition, since the value distribution is defined by the two networks, they are also updated to better represent the value for given data samples. Details for each element in Fig. 4 are described in Sections 2.2 and 2.4.

Since we assume a situation where measurements are uncertain, we add random noise to the observations that serve as the state of the RL agent. In this situation, the state-noisy MDP (SN-MDP) (Sun et al., 2021) is better suited for modeling measurement noise. In the SN-MDP setting, the transition operator is defined as

$$\mathcal{P}^\pi Z(s,a) \overset{D}{:=} Z(s',a'),\ a' \sim \pi(\cdot|\nu(s')),$$

where $\nu(s') \sim N(\cdot|s')$ is the state random variable after the transition under random noise mechanism $N(\cdot|s)$. The corresponding distributional Bellman operator is defined as

$$\mathcal{T}^\pi Z(s,a) \overset{D}{:=} \mathcal{R}(s,a,s') + \gamma\mathcal{P}^\pi Z(s,a).$$

Due to the noisy state observations, the randomness of the four sources (reward, transition dynamics $\mathcal{P}^\pi$, noisy transition N, and the next-state value distribution $Z(s',a')$) is reflected in the SN-MDP setting. One of the contributions of Sun et al. (2021) is that the contraction and convergence properties of the distributional Bellman operator (2) are still maintained in the SN-MDP setting. This finding enables us to deploy the conventional DRL method described in Section 2.3 even with the noisy state observations.

The basic procedure for approximating the value distribution is the same as described in Section 2.4. One difference is that our method adopts a hierarchical structure in contrast to other DRL methods. To deal with multiple actions from the hierarchical policies, we expand the action a in (4) to include two actions a_d and a_c. Therefore, the temporal difference error (4) is modified as follows:

$$\delta_{ij}^t = r_t + \gamma\left[F_{Z',\psi}^{-1}(\hat{\tau}_i) - \sum_{a_d'\in\mathcal{A}^d}\pi^d(a_d'|s')\frac{\beta}{|a_d|}\log\pi^c(a_c'|s',a_d')\right] - F_{Z,\psi}^{-1}(\hat{\tau}_j).$$

In terms of the Bellman operator, since π^d outputs a probability for discrete actions, we can directly compute the expectations to fully

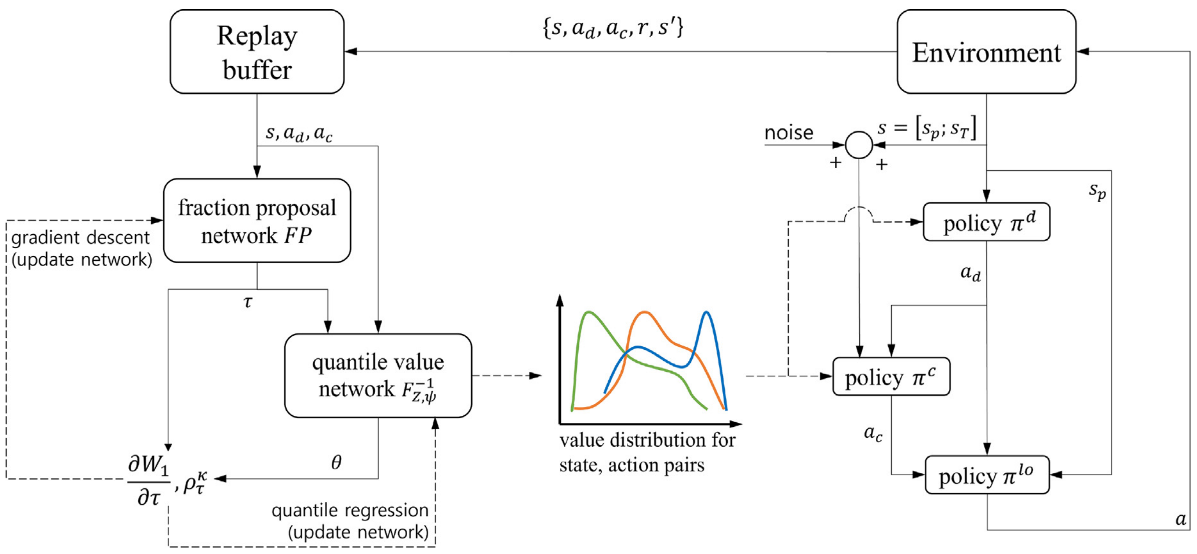

Replay buffer
$\{s, a_d, a_c, r, s'\}$
Environment
s, a_d, a_c
noise
$s = [s_p; s_T]$
+
+
fraction proposal network FP
gradient descent (update network)
s_p
policy π^d
a_d
τ
quantile value network $F^{-1}_{Z,\psi}$
value distribution for state, action pairs
policy π^c
a_c
$\frac{\partial W_1}{\partial \tau}, \rho^\kappa_\tau$
θ
policy π^{lo}
quantile regression (update network)
a

Fig. 4. The overall procedure of our method. The left-hand side of the figure illustrates updating the two networks to approximate the value distribution based on the data collected in the replay buffer. The right-hand side of the figure illustrates the hierarchical structure of the policies. High-level policies are updated based on the approximated value distribution.

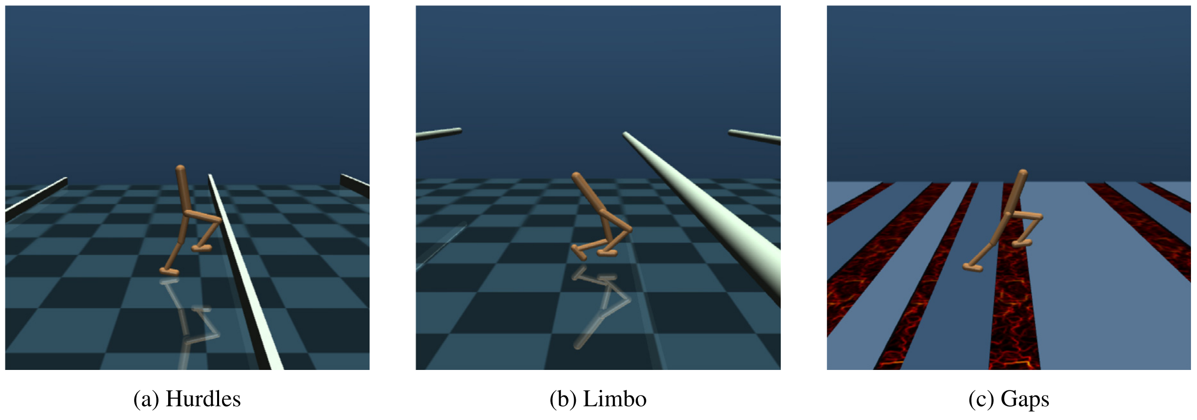


Fig. 5. Examples of benchmark environments with a bipedal robot. For (a) and (b), the spacing and height of obstacles are randomly sampled per episode. In the case of (c), the width and gap of the platform are also randomly sampled per episode.

recover the action distribution. Therefore, the distributional Bellman operator (3) is modified as follows:

$$\mathcal{T}^\pi Z(s, a_d, a_c) \overset{D}{:=} \mathcal{R}(s, a_d, a_c) + \gamma \left[Z(s', a'_d, a'_c) - \sum_{a'_d \in \mathcal{A}^d} \pi^d(a'_d|s') \frac{\beta}{|a_d|} \log \pi^c(a'_c|s', a'_d) \right]. \quad (5)$$

In what follows, we present the convergence properties of our DHRL algorithm.

Lemma 3.1. *$\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ is a γ-contraction in $\bar{d}_p$.*

Lemma 3.2 (*Policy Evaluation*)**.** *For any mapping $Z^0 : \mathcal{S} \times \mathcal{A}^d \times \mathcal{A}^c \to \mathcal{Z}, Z^{k+1} = T^\pi Z^k$ will converge to Z^π as $k \to \infty$.*

Lemma 3.3 (*Policy Improvement*)**.** *Let $\pi_{old} \in \Pi$ and π_{new} as the solution of the minimization problem defined in (1). We have $Q^{\pi_{old}}(s, a_d, a_c) \leq Q^{\pi_{new}}(s, a_d, a_c)$ for all $(s, a_d, a_c) \in \mathcal{S} \times \mathcal{A}^d \times \mathcal{A}^c$, where $Q^\pi(s, a_d, a_c) = \mathbb{E}\left[Z^\pi(s, a_d, a_c)\right]$.*

Proofs of Lemmas 3.1, 3.2, and 3.3 are presented in Appendix. The policies are updated through a policy iteration that consists of 2 steps: a policy evaluation step and a policy improvement step. By iteratively applying the distributional Bellman operator (5) for a given policy π, the value distribution Z^π can be obtained. Once the value distribution is obtained, the policies are improved to achieve higher values by minimizing (1). The policy loss is given as:

$$\mathbb{E}_{s\sim B}\left[\sum_{a_d \in \mathcal{A}^d} \pi^d(a_d|s) \left[\frac{\beta}{|a_d|} \log \pi^c(a_c|s, a_d) - Q(s, a_d, a_c)\right]\right],$$

where B is a replay buffer and $Q(s, a_d, a_c)$ is the action-value function obtained by taking expectation as

$$Q(s, a_d, a_c) = \sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) F^{-1}_{Z,\psi}(\hat{\tau}_i).$$

4. Results: Bipedal robot learning

4.1. Problem formulation

The MuJoCo physics simulator was selected for our benchmark evaluation due to its extensive use in the development of RL algorithms (Schulman et al., 2017; Gehring et al., 2021; Lee and Choi, 2022; Todorov et al., 2012; Ma et al., 2020). In particular, we evaluate our method with a bipedal robot in various MuJoCo task environments implemented with OpenAI Gym (see Fig. 5). In the benchmark

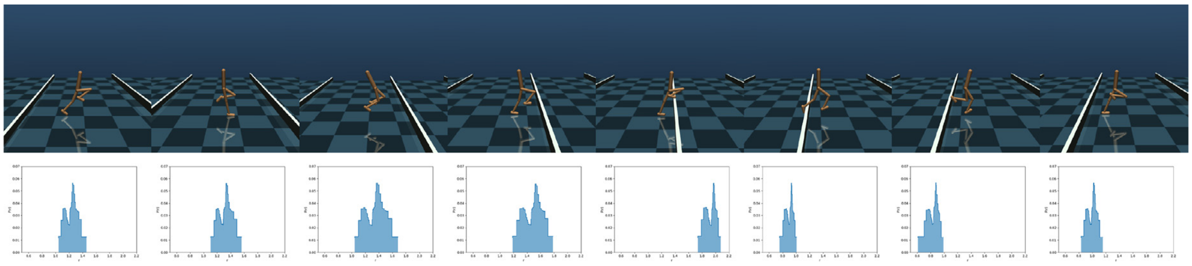


Fig. 6. Predicted value distribution over eight consecutive frames in one of the benchmark environments. The X-axis and Y-axis denote a predicted reward and the probability of receiving that reward, respectively.

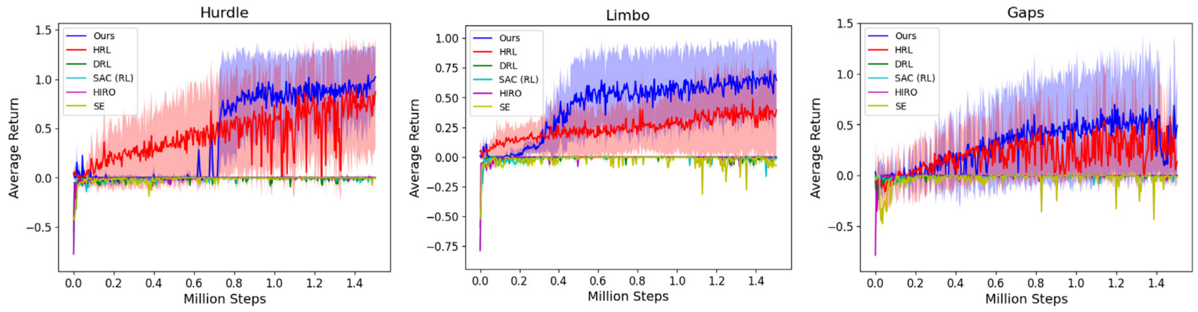


Fig. 7. Training results compared to the state-of-the-art RL algorithms with noisy state observations in three benchmark environments.

environments, observations s include proprioceptive robot states s_p and task-specific observations s_T, i.e., $s = [s_p; s_T]$. In Fig. 5(a), the task-specific observations are the distance to the next hurdle and its height. Similarly, in Fig. 5(b), the task-specific observations are the distance to the next bar and its height. In Fig. 5(c), the task-specific observations are the distances to the next gap and the next platform. The movement of the agent (a bipedal robot) is constrained in the XZ plane. Therefore, feature space $\mathcal{A}^d$ of the robot consists of 5 features: the robot's translation along the X-axis, its position on the Z-axis, the torso rotation around the Y-axis, and the positions of each foot relative to the body. By combining these features, $2^5 - 1 = 31$ combinations are available. The high-level policy π^d uses the observations s as an input and selects one of the 31 combinations a_d. The second-level policy π^c also takes s as an input, and outputs the target configuration a_c for each feature consisting of 7 continuous values: torso position along the X and Z axes, torso rotation around the Y-axis, left foot position along the X and Z axes, and right foot position along the X and Z axes. Finally, the pre-trained low-level policy π^{lo} takes s_p as an input from the observations except for the task-specific observations s_T and passes an action directly to the agent to achieve the target configuration. We use the pre-trained policy from Gehring et al. (2021).

In our simulation, we generate noise and exogenous perturbations that can often occur in real-world physical systems. For example, in a robotics application, sensor readings may be subject to sensor noise or uncertainties, leading to noisy observations of the robot's state. Exogenous perturbations, such as sudden changes in wind or temperature in a robotics application, can cause changes to the reward structure. For example, a previously rewarding action may become unrewarding due to changes in the environment. For the case of noisy state observations, we have added random noise to the task-specific observations, $\hat{p} = p+e$, where p, $\hat{p}$, and e denote the observations, the noisy observations, and the random noise, respectively. e is realized from the α-weighted uniform distribution $\alpha U(-p, p)$, where $U(-p, p)$ generates random variables between $-p$ and p. In addition, we implement exogenous perturbations as external forces acting on the agent's body, which can be viewed as a wind gust. Details of generating perturbations are described in Section 4.4.

4.2. Prediction results of the value distribution

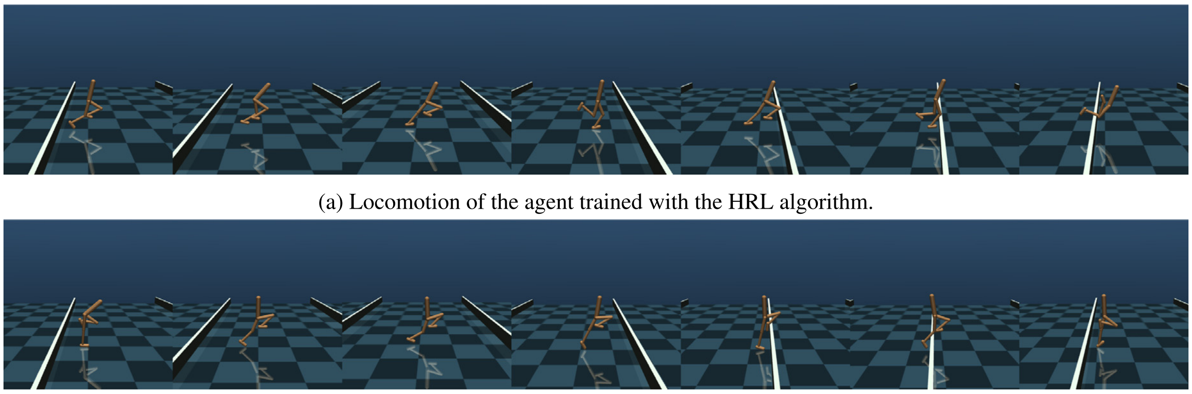
Fig. 6 shows a sequence of predicted value distributions during the robot jumps over hurdles. The X-axis and Y-axis denote the reward and the probability of receiving that reward, respectively. The task is intended to have a sparse reward, and the robot receives an immediate reward of 1 after the robot passes an obstacle and −1 when the robot falls down. The figure clearly indicates the randomness of the environment, with its distribution being widely spread near obstacles and sharp otherwise. Widely spread rewards mean that the robot can receive high or low rewards, while sharp rewards mean that the robot receives relatively constant rewards. When the robot is about to pass the hurdle, it can receive an immediate reward of 1 if it passes the hurdle and −1 if it falls down. Therefore, the expected reward is distributed from low to high values. On the other hand, when the robot is far from the hurdle, the expected reward is relatively constant since the robot receives a constant immediate reward of 0. In addition, since the robot receives a positive reward after passing through the hurdle, the distribution shifts in the X-axis direction as it gets closer to the obstacle.

4.3. Robustness to noisy state observations

In this section, we evaluate the robustness of our method against noise in the state observations. In particular, we first present the ablation study in Section 4.3.1 and gait analysis in Section 4.3.2.

4.3.1. Ablation study

Fig. 7 shows the effectiveness of our method via ablation studies that solve complex sparse-reward tasks with noisy state observations. During training, each algorithm was evaluated at every 4000 steps, and the average return received by the agent was plotted in the figure. Our method outperforms the state-of-the-art RL algorithms alone. The performance of each method is summarized in Table 1. Each number in Table 1 is calculated by averaging the number of successes over 50 evaluations with a fixed policy that has been trained once. Algorithms for HRL and DRL are adopted from Gehring et al. (2021) and Ma et al. (2020), respectively. SAC (Haarnoja et al., 2018) is used as the basic RL algorithm. To provide a more comprehensive comparison, we have applied two additional algorithms, HIerarchical Reinforcement learning



(a) Locomotion of the agent trained with the HRL algorithm.

(b) Locomotion of the agent trained with our method.

Fig. 8. Comparing the locomotion of the agent during one of the benchmark environments.

Table 1
The average number of successes over 50 evaluations with the trained policy across benchmark environments with noisy state observations. The number of successes is defined as the number of obstacles the agent passes through minus the number of falls.

	Hurdles	Limbo	Gaps
Ours	9.26	7.10	2.96
HRL	4.94	2.78	0.76
DRL	−0.02	0.00	0.00
RL	−0.18	−0.08	−0.06

Table 2
The average number of successes over 50 evaluations with the trained policy across benchmark environments with non-stationary exogenous perturbations. The number of successes is defined as the number of obstacles the agent passes through minus the number of falls.

	Hurdles	Limbo	Gaps
Ours	11.38	8.34	3.50
HRL	7.92	6.70	1.48

with Off-policy correction (HIRO) (Nachum et al., 2018) and Switching Ensemble (SE) (Nachum et al., 2019). These methods were also used as benchmarks in Gehring et al. (2021). Solid lines and light-colored areas denote average returns and standard deviations, respectively. In all examples in Fig. 7, DRL and RL agents show no training potential. In the first two examples, the average return of our method exceeds that of HRL after about 700,000 steps and 300,000 steps, respectively. In these two examples, the agent is trained from scratch. However, in the other example, the agent was not trained no matter which algorithm was used. Therefore, we use the transfer learning technique to let the agent start training from the model we saved in the first example. The model used for transfer learning is the model saved at 1,000,000th step of the first example. In conclusion, our method outperforms the state-of-the-art RL algorithms and shows effectiveness in solving sparse-reward tasks with uncertain measurements.

4.3.2. *Gait analysis of RL agents after training*

Fig. 8 shows a sequence of frames in one of the benchmark environments. The agent in Fig. 8(a) is trained with the HRL algorithm and the agent in Fig. 8(b) is trained with our method. When we look closely at the movement of the legs, the agent in Fig. 8(a) moves the foot relatively low before getting close to the hurdle. On the other hand, the agent in Fig. 8(b) moves its foot high almost everywhere. The locomotion in Fig. 8(a) can be viewed as more efficient as long as there is no measurement noise in the observations. However, in situations where measurements are uncertain, the conservative behavior in Fig. 8(b) enables the agent to pass through more hurdles without failure as shown in Fig. 7.

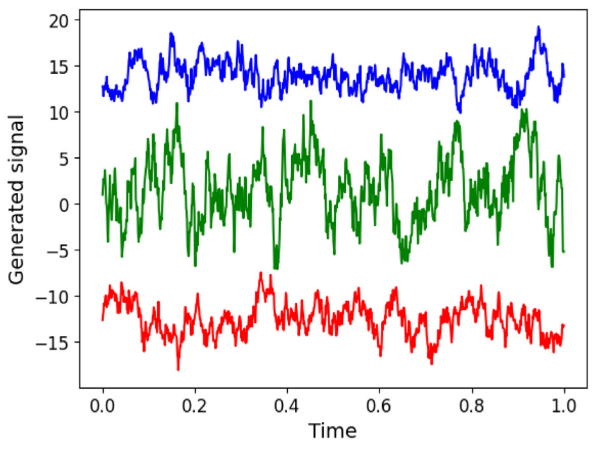
4.4. *Robustness to non-stationary exogenous perturbations*

In this section, we evaluate our method with exogenous perturbations. In real-world physical systems, exogenous perturbations such as wind gusts occur independently of the agent's behavior and evolve randomly. We generate a wind gust in our simulation, as shown in Fig. 9(b). We first pass white noise through a low-pass filter (Gavriluta et al., 2012) and add a randomly selected mean value (positive, negative, and zero). We then apply an external force proportional to the generated signal to the agent's body in the simulator. To ensure generality, we randomly select a new mean value for every 1,000 steps, resulting in a non-stationary stochastic process. Examples of the generated signals with time-varying means to simulate non-stationary wind gusts, are shown in Fig. 9(a).

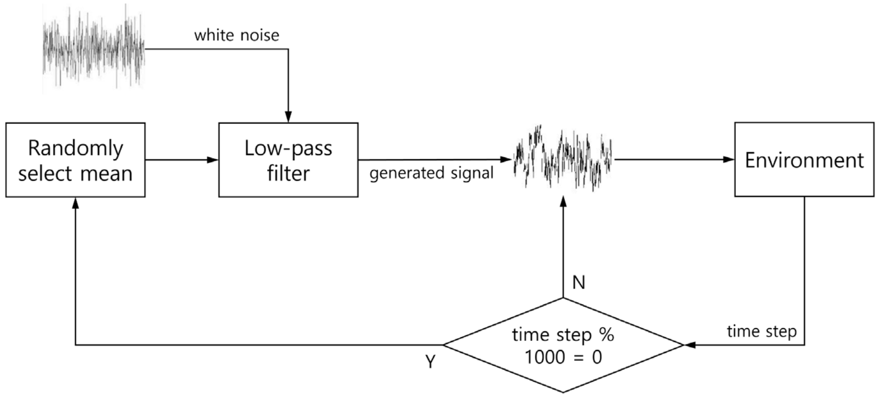
If we add perturbations from scratch during training, the perturbations prevent the agent from collecting useful data samples and thus prevent the agent from learning the policy. Therefore, we adopt a transfer learning technique to ensure that the agent has minimal ability to collect useful data. The pre-trained model used for transfer learning is the model saved at the 800,000th, 300,000th, and 300,000th step of each task in Fig. 7, respectively. The simulation results with non-stationary exogenous perturbations are shown in Fig. 10 and the performance of each algorithm is summarized in Table 2. Fig. 10 shows the average return received by the agent of each algorithm evaluated at every 4000 steps. Each number in Table 2 is calculated by averaging the number of successes over 50 evaluations with a fixed policy that has been trained once. Our method shows a more robust performance than the HRL algorithm. We can conclude that our method is robust to non-stationary exogenous perturbations.

4.5. *Robustness to noisy state observations and non-stationary exogenous perturbations*

In this section, we evaluate our approach with noisy state observations and non-stationary exogenous perturbations applied simultaneously during training. As same as experiments in Section 4.4, we use the transfer learning technique. The results are shown in Fig. 11 and the performance of each algorithm is summarized in Table 3. Fig. 11 shows the average return received by the agent of each algorithm evaluated at every 4000 steps. Each number in Table 3 is calculated by averaging the number of successes over 50 evaluations with a fixed policy that has been trained once. In the first two examples in Fig. 11, our method performs much better than the HRL algorithm. However, in the last example in Fig. 11, both methods show similar performance. We postulate that this result originated from the nature of the tasks. As shown in Fig. 8, the agent trained with our method tends to behave



(a) Examples of three different simulated wind signals. Those are generated by passing white noise sequences with different means and variances through a low-pass filter. One of them is randomly applied to the environment during training.



(b) Wind is simulated by passing white noise through a low-pass filter. For every 1,000 steps, we randomly select mean of the perturbations to generate the non-stationary random processes (e.g., wind gusts changing directions).

Fig. 9. Simulation process and examples of non-stationary exogenous perturbations.

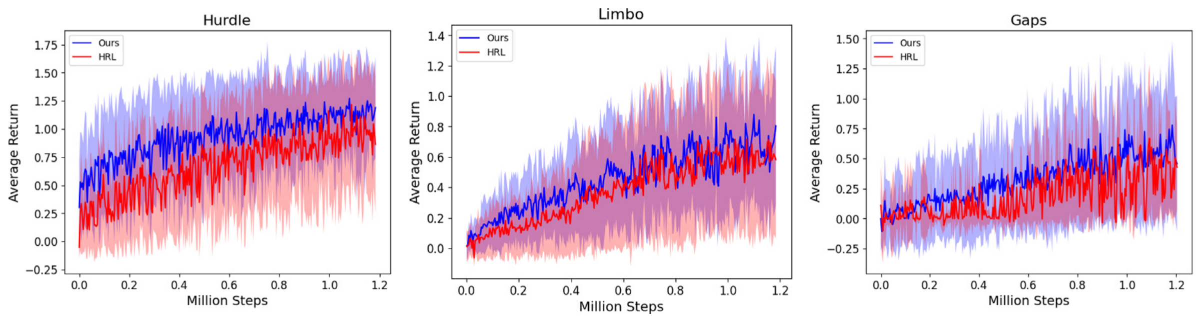


Fig. 10. Training results compared to the HRL algorithm with non-stationary exogenous perturbations in three benchmark environments.

conservatively. For example, in the ‘hurdles’ task, the agent moves its feet high and early before reaching a hurdle. In the ‘limbo’ task, the agent leans its torso low and early before reaching a bar. These conservative behaviors may be helpful in achieving robust performance against noisy observations and exogenous perturbations. On the other hand, in the ‘gaps’ task, since the episode terminates immediately after the agent steps between platforms, this task requires relatively more sophisticated motions than the other tasks. Therefore, conservative behaviors are not very effective in this task, and the average returns of both methods are also lower than the average returns in the other tasks.

5. Results: Autonomous driving of electrical vehicles

5.1. Problem formulation

Autonomous driving is an emerging technology that promises to revolutionize the transportation industry by reducing accidents, improving

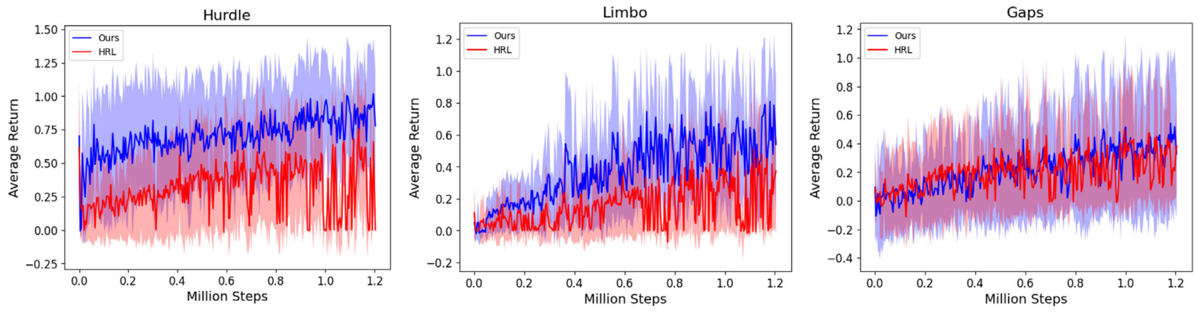


Fig. 11. Training results compared to the HRL algorithm with noisy state observations and non-stationary exogenous perturbations together in three benchmark environments.

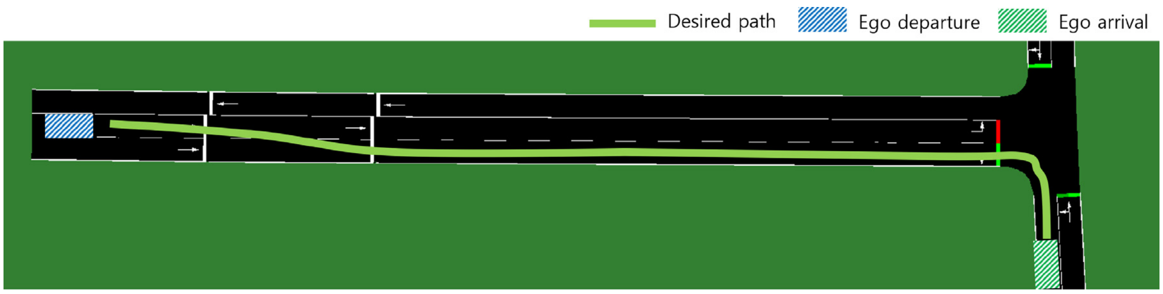


Fig. 12. An example scenario with two lanes. The ego vehicle starts in the left lane and have to change lanes once to reach the target location.

Table 3
The average number of successes over 50 evaluations with the trained policy across benchmark environments with noisy state observations and non-stationary exogenous perturbations. The number of successes is defined as the number of obstacles the agent passes through minus the number of falls.

	Hurdles	Limbo	Gaps
Ours	8.98	5.72	1.60
HRL	5.42	4.88	0.64

Table 4
The averaged absolute acceleration of the vehicle over 50 evaluations during tests.

	Ours	HRL
w/o noise	0.358	1.389
w/ noise	0.409	1.519

traffic flow, and increasing energy efficiency. One of the key challenges in autonomous driving is to design a control system that can handle the uncertainties and variations present in real-world driving conditions. We apply our method to electric autonomous vehicles using the SUMO traffic simulator (German Aerospace Center, Cologne, Germany). The SUMO simulator allows us to easily build scenario maps, and provides a sufficient number of APIs for accessing data. The RL agent's state information consists of three key elements: vehicle state, lane information, and traffic light state. The vehicle state comprises the velocity of the ego vehicle, as well as the distance and velocity of vehicles in the front, left lane, and right lane of the ego vehicle. Lane information includes the current lane index, as well as whether the left or right lane was available. Finally, the traffic light state includes the distance to the next traffic light, whether the next traffic light was green, and the time remaining until it turns green.

In order to control a vehicle, it requires two inherently independent control inputs: steering and throttle. Therefore, we apply our method and the HRL method having hierarchical policies that can generate multi-level actions. Specifically, we define the action space for discrete actions $\mathcal{A}^d$ as follows: change to the left lane, change to the right lane, and keep the current lane. Similarly, we define the action space for continuous actions $\mathcal{A}^c$ as the desired velocity.

We design a 2-lane scenario as shown in Fig. 12. The ego vehicle starts in the left lane and has to change lanes once to reach the target location. Compared to the previous simulation where obstacles are fixed, this scenario is a dynamic environment in which surrounding vehicles are moving along with the ego vehicle. Therefore, it is similar to the case discussed in Section 4.4, where the relative distances to obstacles are varying due to exogenous perturbations.

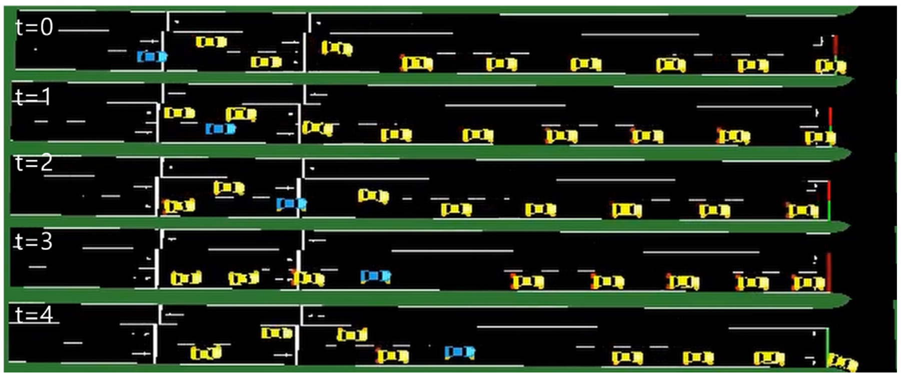
The reward function consists of seven terms representing traveled distance, whether the vehicle is stopped, violation of traffic light signal, control smoothness, battery consumption, collision, and reaching a goal. The contribution of each term is adjusted by its weight.

$$\begin{aligned} r_{total} =& w_{dist} \cdot r_{dist} + w_{stop} \cdot r_{stop} + w_{tls} \cdot r_{tls} + w_{control} \cdot r_{control} \\ &+ w_{battery} \cdot r_{battery} + w_{collision} \cdot r_{collision} + w_{goal} \cdot r_{goal} \end{aligned}$$

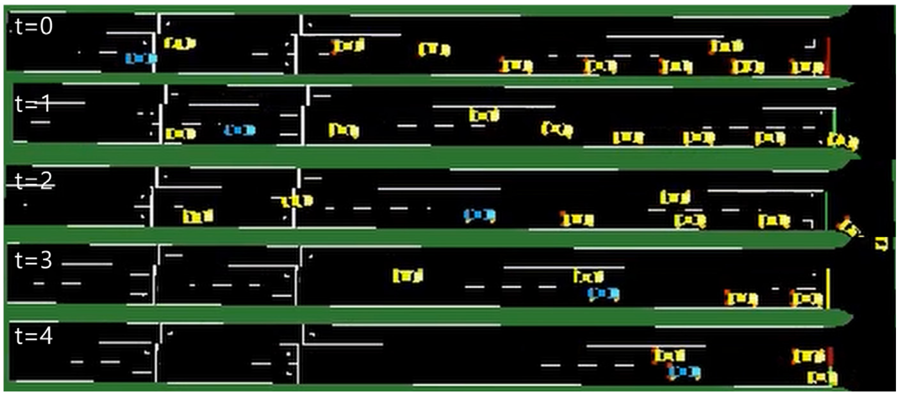
5.2. Training results

We evaluated our method and HRL algorithm in two environments, one without noise and one with noise in observations. Random noise is added to the distance and velocity of vehicles in the front, left lane, and right lane of the ego vehicle, and the distance to the traffic light. The performance of each algorithm is summarized in Table 4. Each number in Table 4 denotes the averaged absolute acceleration of the vehicle over 50 evaluations during tests. To ensure safety, emergency braking is applied to avoid collisions when the distance from the vehicle in front is less than a certain threshold. The results show that our method outperforms HRL in both environments, achieving a lower averaged absolute acceleration of the vehicle so that it simultaneously improves ride comfort and optimizes battery consumption (see Fig. 14).

The agent trained with our method moves relatively slowly to secure a more safe distance from the vehicle in front by predicting uncertain behaviors of neighboring vehicles (see Fig. 13). Therefore, the agent can easily avoid a collision even if the vehicle in front slows down or another vehicle cuts in. In addition, it is robust to measurement



(a) Snapshots of autonomous driving by the agent (blue) trained by DHRL algorithm.



(b) Snapshots of autonomous driving by the agent (blue) trained by HRL algorithm.

Fig. 13. Comparison of agent behaviors in a dynamic environment in which surrounding vehicles are moving. The agent trained with our method moves slowly to maintain a safe distance from the vehicle in front, allowing it to avoid collisions even if the vehicle in front slows down or another vehicle cuts in.

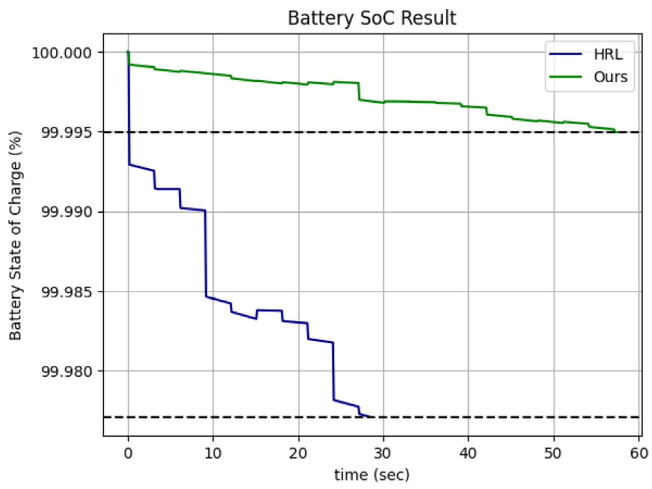


Fig. 14. Comparison of battery consumption between the agent trained with our method (green line) and the agent trained with the HRL method (blue line). The agent trained with our method shows a smoother driving pattern, leading to less battery consumption, while the agent trained with the HRL method repeats acceleration and abrupt braking, resulting in higher battery consumption.

noise. In contrast, the agent trained with HRL moves relatively quickly. Therefore, the agent barely avoids a collision by hard braking when the vehicle in front slows down, or it is difficult to avoid a collision when another vehicle cuts in.

6. Conclusions

This paper introduces DHRL, a novel approach that combines HRL and DRL techniques to address complex sparse-reward tasks with uncertainties in observations and non-stationary exogenous perturbations. Our algorithm's effectiveness is demonstrated through analysis of its convergence properties and evaluation of its performance in a variety of challenging MuJoCo bipedal robot learning environments. We also apply DHRL to an autonomous electric vehicle navigating urban streets and show that it outperforms HRL in terms of ride comfort and battery consumption. Our approach is successful in handling challenging real-world applications where uncertainties due to measurement noise and exogenous perturbations are present. By modeling random rewards as random variables following a value distribution, our algorithm exhibits robust performance in dealing with uncertainties. Additionally, the hierarchical policy structure of DHRL allows for the handling of more complex tasks compared to DRL.

While promising, there are limitations to the effectiveness of our approach. For example, as depicted in Fig. 8, DHRL produces a more conservative agent with higher foot movements, resulting in more robust performance but less effective for tasks requiring precise control. In future work, we plan to extend the applicability of our method to other applications such as robotic manipulators and drones.

Overall, our proposed DHRL algorithm provides a basis for the development of more robust and efficient RL algorithms capable of handling complex tasks under harsh and uncertain real-world conditions.

CRediT authorship contribution statement

Jehyun Park: Conceptualization, Methodology, Writing – original draft. **Jongeun Choi:** Writing – review & editing, Supervision. **Sungjae Nah:** Software. **Dohee Kim:** Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2B5B01002620) and by Hyundai Motor Company.

Appendix

In this section, we provide proofs of lemmas we presented in Section 3 and pseudo code of the algorithm.

Lemma 3.1. *$\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ is a γ-contraction in $\bar{d}_p$.*

Proof. For any $(s, a_d, a_c) \in \mathcal{S} \times \mathcal{A}^d \times \mathcal{A}^c$, using the properties of the Wasserstein metric (Bellemare et al., 2017), we have

$$\begin{aligned}
&W_p(T^\pi Z_1(s,a_d,a_c), T^\pi Z_2(s,a_d,a_c))\\
&= W_p\Bigg(\mathcal{R}(s,a_d,a_c) + \gamma\left[Z_1(s',a'_d,a'_c) - \sum_{a'_d\in\mathcal{A}^d} \pi^d(a'_d|s')\frac{\beta}{|a_d|}\log\pi^c(a'_c|s',a'_d)\right],\\
&\qquad \mathcal{R}(s,a_d,a_c) + \gamma\left[Z_2(s',a'_d,a'_c) - \sum_{a'_d\in\mathcal{A}^d} \pi^d(a'_d|s')\frac{\beta}{|a_d|}\log\pi^c(a'_c|s',a'_d)\right]\Bigg)\\
&\le \gamma W_p(Z_1(s',a'_d,a'_c), Z_2(s',a'_d,a'_c))\\
&\le \gamma \sup_{s',a'_d,a'_c} W_p(Z_1(s',a'_d,a'_c), Z_2(s',a'_d,a'_c)).
\end{aligned}$$

By the definition of $\bar{d}_p$,

$$\begin{aligned}
&\bar{d}_p(T^\pi Z_1, T^\pi Z_2)\\
&= \sup_{s',a'_d,a'_c} W_p\left(T^\pi Z_1(s,a_d,a_c), T^\pi Z_2(s,a_d,a_c)\right)\\
&\le \gamma \sup_{s',a'_d,a'_c} W_p(Z_1(s',a'_d,a'_c), Z_2(s',a'_d,a'_c))\\
&= \gamma \bar{d}_p(Z_1, Z_2). \quad \square
\end{aligned}$$

Lemma 3.2 (*Policy Evaluation*). *For any mapping $Z^0 : \mathcal{S}\times\mathcal{A}^d\times\mathcal{A}^c \to \mathcal{Z}$, $Z^{k+1} = T^\pi Z^k$ will converge to Z^π as $k \to \infty$.*

Proof. As we have proved that T^π is a γ-contraction, policy evaluation can be obtained by repeatedly applying T^π. $\square$

Lemma 3.3 (*Policy Improvement*). *Let $\pi_{old} \in \Pi$ and π_{new} as the solution of the minimization problem defined in (1). We have $Q^{\pi_{old}}(s,a_d,a_c) \le Q^{\pi_{new}}(s,a_d,a_c)$ for all $(s,a_d,a_c) \in \mathcal{S}\times\mathcal{A}^d\times\mathcal{A}^c$, where $Q^\pi(s,a_d,a_c) = \mathbb{E}[Z^\pi(s,a_d,a_c)]$.*

Proof. First, we define the soft action-value for an arbitrary policy $\pi \in \Pi$ as the expectation of the value distribution:

$$\begin{aligned}
&Q^\pi(s,a_d,a_c) = \mathbb{E}[Z^\pi(s,a_d,a_c)]\\
&= \mathbb{E}[\mathcal{R}(s,a_d,a_c)]\\
&+\gamma \mathop{\mathbb{E}}_{\substack{s'\sim\mathcal{P}(\cdot|s,a_d,a_c)\\ a'_d\sim\pi^d(\cdot|s')\\ a'_c\sim\pi^c(\cdot|s',a'_d)}}\left[Z^\pi(s',a'_d,a'_c) - \sum_{a'_d\in\mathcal{A}^d}\pi^d(a'_d|s')\frac{\beta}{|a'_d|}\log\pi^c(a'_c|s',a'_d)\right].
\end{aligned}$$

We then define a minimization problem modifying the Eq. (1) as follows:

$$\pi_{new}(\cdot|s) = \arg\min_{\pi'\in\Pi} D_{KL}\left(\pi'(\cdot|s) \,\|\, \frac{\exp\left(\frac{|a_d|}{\beta}Q^{\pi_{old}}(s,\cdot)\right)}{\triangle^{\pi_{old}}}\right).$$

We can choose π_{new} as the solution to the aforementioned minimization problem, we have

$$\begin{aligned}
&\mathop{\mathbb{E}}_{a_c\sim\pi^c_{new}(\cdot|s,a_d)}\Bigg[\sum_{a_d\in\mathcal{A}^d}\pi^d_{new}(a_d|s)\log\pi^c_{new}(a_c|s,a_d)\\
&\qquad -\frac{|a_d|}{\beta}Q^{\pi_{old}}(s,a_d,a_c) + \log\triangle^{\pi_{old}}(s)\Bigg]\\
&\le \mathop{\mathbb{E}}_{a_c\sim\pi^c_{old}(\cdot|s,a_d)}\Bigg[\sum_{a_d\in\mathcal{A}^d}\pi^d_{old}(a_d|s)\log\pi^c_{old}(a_c|s,a_d)\\
&\qquad -\frac{|a_d|}{\beta}Q^{\pi_{old}}(s,a_d,a_c) + \log\triangle^{\pi_{old}}(s)\Bigg].
\end{aligned}$$

The partition function $\triangle^{\pi_{old}}$ can be ignored since it only depends on the state, and does not contribute to the gradient for the new policy. Then the inequality reduced as follows:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{a_c\sim\pi^c_{new}(\cdot|s,a_d)}\left[\sum_{a_d\in\mathcal{A}^d}\pi^d_{new}(a_d|s)\frac{\beta}{|a_d|}\log\pi^c_{new}(a_c|s,a_d) - Q^{\pi_{old}}(s,a_d,a_c)\right]\\
&\le \mathop{\mathbb{E}}_{a_c\sim\pi^c_{old}(\cdot|s,a_d)}\left[\sum_{a_d\in\mathcal{A}^d}\pi^d_{old}(a_d|s)\frac{\beta}{|a_d|}\log\pi^c_{old}(a_c|s,a_d) - Q^{\pi_{old}}(s,a_d,a_c)\right].
\end{aligned}$$

Using the above inequality to repeatedly expand Q^π of the soft Bellman equation, the soft Bellman equation becomes

$$\begin{aligned}
&Q^{\pi_{old}}\left(s^t,a^t_d,a^t_c\right)\\
&= \mathbb{E}\left[\mathcal{R}\left(s^t,a^t_d,a^t_c\right)\right] + \gamma \mathop{\mathbb{E}}_{\substack{s^{t+1}\sim\mathcal{P}(\cdot|s^t,a^t_d,a^t_c)\\ a^{t+1}_d\sim\pi^d_{old}(\cdot|s^{t+1})\\ a^{t+1}_c\sim\pi^c_{old}(\cdot|s^{t+1},a^{t+1}_d)}}\Bigg[Q^{\pi_{old}}\left(s^{t+1},a^{t+1}_d,a^{t+1}_c\right)\\
&\qquad - \sum_{a_d\in\mathcal{A}^d}\pi^d_{old}(a_d|s)\frac{\beta}{|a_d|}\log\pi^c_{old}\left(a^{t+1}_c|s^{t+1},a^{t+1}_d\right)\Bigg]\\
&\le \mathbb{E}\left[\mathcal{R}\left(s^t,a^t_d,a^t_c\right)\right] + \gamma \mathop{\mathbb{E}}_{\substack{s^{t+1}\sim\mathcal{P}(\cdot|s^t,a^t_d,a^t_c)\\ a^{t+1}_d\sim\pi^d_{new}(\cdot|s^{t+1})\\ a^{t+1}_c\sim\pi^c_{new}(\cdot|s^{t+1},a^{t+1}_d)}}\Bigg[Q^{\pi_{old}}\left(s^{t+1},a^{t+1}_d,a^{t+1}_c\right)\\
&\qquad - \sum_{a_d\in\mathcal{A}^d}\pi^d_{new}(a_d|s)\frac{\beta}{|a_d|}\log\pi^c_{new}\left(a^{t+1}_c|s^{t+1},a^{t+1}_d\right)\Bigg]\\
&\vdots\\
&\le Q^{\pi_{new}}(s^t,a^t_d,a^t_c).
\end{aligned}$$

Since the Bellman operator $T^\pi Z$ converges to Z^π according to Lemma 3.2, the right-hand side of the above equation converge to Q^{new}. $\square$

Algorithm

Algorithm 1 Training Procedure

```
**Require:** Low-level policy $\pi^{lo}$, high-level action interval $c$
Initialize policies $\pi^d, \pi^c$, quantile value network $F^{-1}_{Z,\psi}$, fraction proposal network $FP$
$B$ = replay buffer
$s \sim S_0, t = 0$
**for** training step **do**
  **if** t MOD c == 0 **then**
    $a_d \sim \pi^d(s)$
    $a_c \sim \pi^c(s, a_d)$
  **end if**
  $a = \pi^{lo}(a|s_p, a_d, a_c)$
  $s', r$ = ENV.STEP$(s, a_d, a_c)$
  B.append$(s, a_d, a_c, r, s')$
  Update $a_c$
  $s = s'$
  $t = t + 1$
  **if** update interval **then**
    **for** gradient step **do**
      Sample batch from $B$, compute losses
      Update $F^{-1}_{Z,\psi}$, FP, $\pi^d, \pi^c$, $\beta$
    **end for**
  **end if**
**end for**
```

References

Bellemare, M.G., Dabney, W., Munos, R., 2017. A distributional perspective on reinforcement learning. In: International Conference on Machine Learning. PMLR, pp. 449–458.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. Openai gym. arXiv preprint arXiv:1606.01540.

Cho, D., Kim, J., Kim, H.J., 2022. Unsupervised reinforcement learning for transferable manipulation skill discovery. IEEE Robot. Autom. Lett..

Dabney, W., Ostrovski, G., Silver, D., Munos, R., 2018a. Implicit quantile networks for distributional reinforcement learning. In: International Conference on Machine Learning. PMLR, pp. 1096–1105.

Dabney, W., Rowland, M., Bellemare, M., Munos, R., 2018b. Distributional reinforcement learning with quantile regression. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32.

Flet-Berliac, Y., 2019. The promise of hierarchical reinforcement learning, Vol. 9. The Gradient.

Gavriluta, C., Spataru, S., Mosincat, I., Citro, C., Candela, I., Rodriguez, P., 2012. Complete methodology on generating realistic wind speed profiles based on measurements. In: International Conference on Renewable Energies and Power Quality, Vol. 1.

Gehring, J., Synnaeve, G., Krause, A., Usunier, N., 2021. Hierarchical skills for efficient exploration. Adv. Neural Inf. Process. Syst. 34, 11553–11564.

Ghadirzadeh, A., Chen, X., Yin, W., Yi, Z., Björkman, M., Kragic, D., 2020. Human-centered collaborative robots with deep reinforcement learning. IEEE Robot. Autom. Lett. 6 (2), 566–571.

Ha, T., Lee, G., Kim, D., Oh, S., 2021. Road graphical neural networks for autonomous roundabout driving. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 162–167.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al., 2018. Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.

Han, J.I., Lee, J.-H., Choi, H.S., Kim, J.-H., Choi, J., 2022. Policy design for an ankle-foot orthosis using simulated physical human-robot interaction via deep reinforcement learning. IEEE Trans. Neural Syst. Rehabil. Eng..

Huber, P.J., 1992. Robust estimation of a location parameter. In: Breakthroughs in Statistics. Springer, pp. 492–518.

Ji, Y., Li, Z., Sun, Y., Peng, X.B., Levine, S., Berseth, G., Sreenath, K., 2022. Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot. arXiv preprint arXiv:2208.01160.

Kim, M., Lee, S., Lim, J., Choi, J., Kang, S.G., 2020. Unexpected collision avoidance driving strategy using deep reinforcement learning. IEEE Access 8, 17243–17252.

Kim, M., Seo, J., Lee, M., Choi, J., 2021. Vision-based uncertainty-aware lane keeping strategy using deep reinforcement learning. J. Dyn. Syst. Meas. Control 143 (8).

Lee, J.-H., Choi, J., 2022. Hierarchical primitive composition: Simultaneous activation of skills with inconsistent action dimensions in multiple hierarchies. IEEE Robot. Autom. Lett..

Levine, S., Kumar, A., Tucker, G., Fu, J., 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643.

Li, H., Ding, X., 2023. Adaptive and intelligent robot task planning for home service: A review. Eng. Appl. Artif. Intell. 117, 105618.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Lim, J., Ha, S., Choi, J., 2020. Prediction of reward functions for deep reinforcement learning via Gaussian process regression. IEEE/ASME Trans. Mechatronics 25 (4), 1739–1746.

Lowet, A.S., Zheng, Q., Matias, S., Drugowitsch, J., Uchida, N., 2020. Distributional reinforcement learning in the brain. Trends Neurosci. 43 (12), 980–997.

Ma, X., Xia, L., Zhou, Z., Yang, J., Zhao, Q., 2020. DSAC: distributional soft actor critic for risk-sensitive reinforcement learning. arXiv preprint arXiv:2004.14547.

Mavrin, B., Yao, H., Kong, L., Wu, K., Yu, Y., 2019. Distributional reinforcement learning for efficient exploration. In: International Conference on Machine Learning. PMLR, pp. 4424–4434.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. PMLR, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Mohri, M., Rostamizadeh, A., Talwalkar, A., 2018. Foundations of Machine Learning. MIT Press.

Müller, A., 1997. Integral probability metrics and their generating classes of functions. Adv. Appl. Probab. 29 (2), 429–443.

Nachum, O., Gu, S.S., Lee, H., Levine, S., 2018. Data-efficient hierarchical reinforcement learning. Adv. Neural Inf. Process. Syst. 31.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., Levine, S., 2019. Why does hierarchy (sometimes) work so well in reinforcement learning? arXiv preprint arXiv:1909.10618.

Pateria, S., Subagdja, B., Tan, A.-h., Quek, C., 2021. Hierarchical reinforcement learning: A comprehensive survey. ACM Comput. Surv. 54 (5), 1–35.

Pylorof, D., Garcia, H.E., 2022. A reinforcement learning approach to long-horizon operations, health, and maintenance supervisory control of advanced energy systems. Eng. Appl. Artif. Intell. 116, 105454.

Samsonov, V., Hicham, K.B., Meisen, T., 2022. Reinforcement learning in manufacturing control: Baselines, challenges and ways forward. Eng. Appl. Artif. Intell. 112, 104868.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al., 2017. Mastering the game of go without human knowledge. Nature 550 (7676), 354–359.

Smith, L., Kostrikov, I., Levine, S., 2022. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. arXiv preprint arXiv:2208.07860.

Sun, K., Liu, Y., Zhao, Y., Yao, H., Jui, S., Kong, L., 2021. Exploring the robustness of distributional reinforcement learning against noisy state observations. arXiv preprint arXiv:2109.08776.

Todorov, E., Erez, T., Tassa, Y., 2012. Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 5026–5033.

Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30.

Wurman, P.R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T.J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., et al., 2022. Outracing champion gran turismo drivers with deep reinforcement learning. Nature 602 (7896), 223–228.

Yang, D., Zhao, L., Lin, Z., Qin, T., Bian, J., Liu, T.-Y., 2019. Fully parameterized quantile function for distributional reinforcement learning. Adv. Neural Inf. Process. Syst. 32.

Yong, H., Seo, J., Kim, J., Kim, M., Choi, J., 2022. Suspension control strategies using switched soft actor-critic models for real roads. IEEE Trans. Ind. Electron..

Zheng, K., Yang, H., Liu, S., Zhang, K., Lei, L., 2022. A behaviour decision method based on reinforcement learning for autonomous driving. IEEE Internet Things J..