HCIS | Human-centric Computing and Information Sciences

RESEARCH (Research Manuscript) Open Access

Human-centric Computing and Information Sciences (2022) 12:01
DOI: https://doi.org/10.22967/HCIS.2022.12.002
Received: August 6, 2021; Accepted: November 11, 2021; Published: January 15, 2022

Prioritized Environment Configuration for Drone Control with Deep Reinforcement Learning

Sooyoung Jang[1] and Changbeom Choi[2,*]

Abstract

In reinforcement learning, first, the agent collects experiences by interacting with the environment through trial-and-errors (experience collection stage) and then learns from the collected experiences (learning stage). This two-stage training process repeats until the agent solves a given task and requires a lot of experience, computation power, and time for training the agent. Therefore, many studies are conducted to improve the training speed and performance to mitigate them, focusing on the learning stage. This paper focuses on the experience collection stage and proposes a prioritized environment configuration that prioritizes and stochastically samples the effective configuration for initializing the environment for every episode. Therefore, we can provide the environments initialized with the configuration suitable for effective experience collection to the agent. The proposed algorithm can complement the reinforcement learning algorithms that focus on the learning stage. We have shown speed and performance improvement by applying the prioritized environment configuration to an autonomous drone flight simulator. In addition, the results show that the proposed algorithm works well with both on-policy and off-policy reinforcement learning algorithms in distributed framework with multiple workers.

Keywords

Deep Reinforcement Learning, Machine Learning, Prioritized Environment Configuration, Environment Initialization, Drone Control

1. Introduction

Research interests in reinforcement learning are continuously increasing year by year [1]. Accordingly, the range of applications continuously expands to communication [2], abnormal detection [3], privacy information sanitization [4], drone control [5, 6], etc. Furthermore, various services that focus on unrestricted mobility of drones have emerged, such as the internet of drones [7] and drone delivery, thanks to the advances in hardware. They are accelerating the research of drone deep reinforcement learning [5, 6], which we will focus on in this paper. So far, it seems that reinforcement learning is the universal key to solving drone control tasks. However, there are caveats to using reinforcement learning. For example, it is time-consuming, expensive, and unsafe to train drone control agents, especially in the

***Corresponding Author:** Changbeom Choi (cbchoi@hanbat.ac.kr)
[1]Intelligence Convergence Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea
[2]Department of Computer Engineering, Hanbat National University, Daejeon, Korea

real world. As trial-and-error is the basis of reinforcement learning, tremendous and diverse experiences are essential. Experiences with dangerous states and actions are required to obtain generalized high-performance agents to solve the problems. It is more so in model-free reinforcement learning, in which the agents do not know the environment models. This requirement makes the simulator an essential component for training the drone control agents with reinforcement learning. The advantages of the simulator are as follows:

- It is easy to acquire diverse experiences by changing environment configurations such as coordinates, friction coefficient, weather, wind speed, and maps for autonomous drone flight.
- It is relatively easy to improve the training speed through distributed framework.
- It is cost-effective since failing in the simulator does not cost anything.

Reinforcement learning can be divided into two stages. The first stage is to collect experiences through rollouts, and the second stage is to train the policy with those experiences. Accordingly, two questions arise to enhance the training speed and performance: (i) How can we collect effective experiences for the policy update? (ii) How can we effectively learn from those collected experiences?

Research results for the latter include prioritized experience replay (PER) [8] and Ape-X [9] as well as proximal policy optimization (PPO) [10], soft actor-critic (SAC) [11], and many mores.

In the former case, the researches mainly focus on curriculum learning [12]. Curriculum learning provides auxiliary tasks. Similar to human learning, it provides progressively more challenging auxiliary tasks for solving a given task. However, the main goal of curriculum learning is to efficiently solve a given task, not to generalize through learning in various environmental configurations with similar levels of difficulty. There are also challenges in generating adequate auxiliary tasks; bad auxiliary tasks may worsen training.

By predefining environment configurations, we can use the simulator to reproduce various situations to obtain general policies: predefine coordinates sets of various start and goal positions to prevent the agent from memorizing the specific route, predefine various friction coefficients for better sim-to-real transfer, or their combinations. Most of the researchers let the simulator randomly configures them. However, it is inefficient to repeat environment configurations that the agents are good at for both experience collection and training. By prioritizing the environment configurations instead of random sampling, we can enhance the performance of deep reinforcement learning (DRL).

In this paper, we propose prioritized environment configuration (PEC). The proposed algorithm prioritizes the environment configurations, stochastically samples a configuration according to the priority, and initializes the environment with the sampled configuration. Effectively initialized environments can help the agent collect effective experiences. The goal is to improve the training speed and performance of DRL by enabling effective experience collection through the environment initialized with the prioritized configurations. The main contributions of the paper are as follows:

- We proposed a priority metric based on the failures and a stochastic sampling method based on that metric. It allows reflecting the difficulty felt by the agent, i.e., perceived difficulty, furthermore can dynamically adapt to the perceived difficulty of the agent as training progresses without human expertise and intervention. For example, if specific configurations are sampled a lot in the current training iteration, then the failure ratio of those configurations may decrease. It results in the priorities of these settings being suppressed at the following training iteration, enabling efficient training.
- We showed improvements in both training speed and performance on the autonomous drone flight simulator.
- We showed that PEC works well with both on-policy and off-policy DRL algorithms in a distributed framework with multiple workers. Note that PEC complementarily works with reinforcement learning algorithms that focuses on learning from the collected experiences such as PPO [10] and Ape-X DQN [9].

The rest of this paper is organized as follows: Section 2 depicts the related works. Section 3 proposes the prioritized environment configuration algorithm. Section 4 describes the environment that we have utilized to verify and analyze the algorithm, and Section 5 presents the experiment results. Finally,

Section 6 concludes the paper.

2. Related Work

Reinforcement learning requires a lot of experience, computing resources, and learning time. So there have been many studies to increase data efficiency and accelerate training. Among the various works, we will state related works focusing on prioritization, randomization, and curriculum learning, which are highly related to PEC.

Prioritization: DQN [13] first showed human-level performance over many Atari games using deep reinforcement learning. One of the features of DQN is an experience replay which reduces the correlation of the training data, thus, stabilizing the training process. Since the success of DQN, many extensions have been reported, and among them, PER [8] and Ape-X [9] exists.

PER prioritizes the sampling according to temporal-difference error instead of random sampling experiences for learning from the experience replay buffer as DQN does. As a result, it improved both training speed and performance. In addition, Ape-X proposed a distributed architecture that can incorporate PER. Both proved that prioritization which PEC utilizes could yield better training speed and performance than uniform sampling. However, their purpose is to learn from the experiences effectively, whereas PEC is to collect experiences effectively. This difference implies that we can combine PEC and Ape-X. We present the results of PEC with Ape-X in Section 5.5.

Randomization: There are studies on randomization to learn generalized agents for sim-to-real. Two main categories are dynamics randomization and domain randomization: dynamics randomization focuses on randomizing the dynamics of the simulator (e.g., mass, friction, dampings), whereas domain randomization focuses on the domain (e.g., colors, textures, appearance). Dynamics randomization suggested in [14] randomizes the dynamics for the sim-to-real transfer of the robotic arms. In the paper, they configured 95 randomized configurations and randomly sampled them for every episode. Automatic domain randomization is suggested in [15]. It randomizes both domain and dynamics. Moreover, it keeps increasing the boundary of the randomization range as training progresses. The results in [14, 15] both reported the performance of real-world robotic arms similar to that in a simulation. They performed a random selection of the configurations. In contrast, the main contribution of PEC is to prioritize the configurations to avoid unnecessarily many repetitions of easy configurations for efficient training.

Curriculum learning: Curriculum learning provides increasingly complex auxiliary tasks, namely curriculum, to agents to solve a given task. By doing so, the agent can solve a given task that seems impossible. Various types of curriculum learning methods are being studied, which are well organized in [12]. Training speed and performance in environments with high-dimensional continuous action space can be enhanced by progressively increasing the distance of the target point that the agent should reach [16]. GoalGAN [17] proposes automatically generating appropriate goals of intermediate difficulties to solve a task using a generative adversarial network (GAN). In teacher-student curriculum learning [18], the teacher tries to provide the appropriate subtasks to allow the student to solve a complex task. They created five maps with progressive difficulties using Minecraft environments and evaluated the performance with the most complex map. Their method dramatically improved the performance compared to uniform sampling and comparable to the manual curriculum setting. Unlike curriculum learning, which sets auxiliary tasks of various difficulties, the algorithm proposed in this paper aims to effectively utilize various environmental configurations with similar difficulties to learn general agents efficiently. We can utilize curriculum learning and the proposed algorithms together to improve performance. For example, [18] uniformly samples agents and target positions for each map; we can improve the training speed by prioritizing the sampling of the positions using the proposed algorithm.

3. Prioritized Environment Configuration

In this section, we present the motivation, assumption, and the details of the proposed algorithm, PEC.

PEC is motivated by the following questions: (i) Can the environment be initialized by prioritizing configurations for effective experience collection? Can such prioritization (ii) be adaptive to the training progress? (iii) be needless of expert knowledge? (iv) be robust to outliers? (v) be operated in a distributed reinforcement learning framework?

PEC assumes that there is more than one configurable environment configuration in the environment. Autonomous drone flight simulators or autonomous driving simulators [19] usually provide various configurable environment configurations. Examples include coordinates for navigation, navigation maps, weather (e.g., rainy, sunny, cloudy), and wind velocity.

From now on, we present the details of the algorithm.

First, the priority of environment configuration n, p_n is calculated as follows.

$$p_n = \begin{cases} fr_n + \alpha, & if\ t_n > 0 \\ 1, & if\ t_n = 0 \end{cases} \quad (1)$$

Failure ratio, fr_n, is defined as the number of failures of the configuration n, f_n, divided by the number of trials of the configuration n, t_n. $\alpha \geq 0$ is to control the dependency of failure ratios on the priority. It is the only hyperparameter directly related to the algorithm. As α increases, the dependence on the failure ratio decreases diminishing the effect of PEC. Conversely, as α decreases, the dependence on the failure ratio increases, so the sampling is biased towards some specific configurations. By setting the priority as in Equation (1), PEC can provide the sampling probabilities of environment configurations not only with zero trials but also with zero failures with low complexity. It is essential to guarantee the sampling probabilities of successful configurations at some level as the policy might forget them as training progresses.

Then, sampling probability, P_n, is computed as follows. N is the total number of environment configurations.

$$P_n = \frac{p_n}{\sum_{n=1}^{N} p_n} \quad (2)$$

PEC provides stochastic prioritization as it performs prioritized sampling based on the sampling probability. The sampling probability based on Equation (2) has the following advantages.

- *Adaptability to the agent's training progress*: Sampling probability can adapt to changing perceived difficulty levels of the agents for each environment configuration as the training progresses. The perceived difficulty levels of the agents change dynamically depending on the number of trials per environment configuration, which can be confirmed in the experiment results.
- *No need for expert knowledge*: Since PEC automatically configures the sampling probability, there is no need for an expert to set the difficulty levels of environment configurations manually.
- *Robust training*: Since the failure ratio has a value from 0 to 1, outliers, e.g., too big or too small, do not occur, enabling more robust training.
- *Operability with the distributed framework*: PEC can operate with the distributed framework by tracking the number of failures and trials per worker and aggregating them to calculate the sampling probability.
- *Low time complexity*: The time complexity of the proposed algorithm is $O(n)$ due to the proportional sampling from discrete distribution [20], where n is the number of environment configurations.

The procedure of PEC based on the proposed sampling probability in Equation (2) is as follows.

For each training iteration, i:

(Line 2–10 in Fig. 1) Each M parallel worker collects experiences with the sampling probability, P, until they sum to train batch size, TB. Specifically, each worker (1) proportionally samples an

environment configuration, n, based on the sampling probability, P^i (Line 4); (2) initializes an environment, e, with the sampled configuration, n (Line 5); (3) run policy, π^i, and collect experiences until the end of the episode in the environment, e (Line 6); and (4) update the number of failures and trials considering the results of the episode (Line 7–10).

(Line 11 in Fig. 1) The collected experiences are utilized to optimize the policy, π^i. Any DRL algorithm such as PPO or Ape-X DQN can be adopted for policy optimization.

(Line 12–16 in Fig. 1) The sampling probability for the next iteration is computed. First, we aggregate the number of failures, $f^{i,w}$, and trials, $t^{i,w}$, from all workers, w, and update the number of failures, f^i, and t^i (Line 12–13). Then, we compute the sampling probability by Equation (2) based on the updated f^{i+1}, and t^{i+1} (Line 14). Finally, we initialize $f^{i+1,w}$, and $t^{i+1,w}$ to 0 for all workers, w (Line 15–16).

Input

TI: Train iteration, TB: Train batch size, M: Number of workers, α: Dependency control parameter

Initialize

P^0: The sampling probability, f^0: The number of failures, t^0: The number of trials, π^0: policy

Procedure

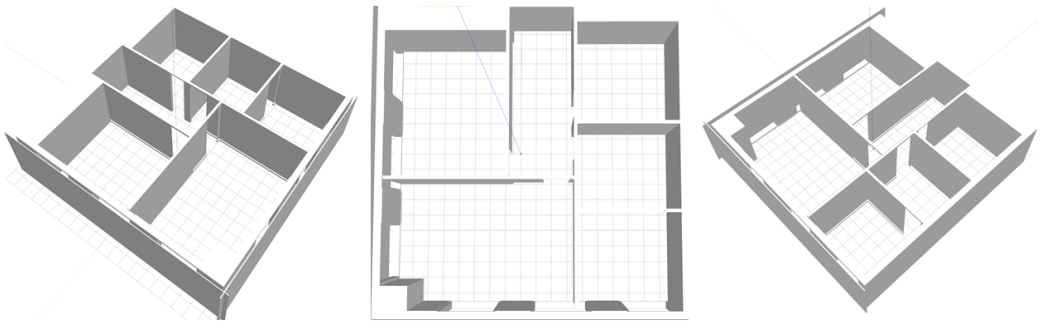
1: **for** $i = 1$ **to** TI **do**
2: **while** collected experiences < TB **do**
3: **for each** $w = 1$ **to** M **do**
4: $n \leftarrow$ ProportionalSampling(P^i)
5: $e \leftarrow$ InitializeEnvironment(n)
6: Run policy π^i and collect experiences until episode end in environment e
7: **if** episode failed **then**
8: $f_n^{i,w} \leftarrow f_n^{i,w} + 1$
9: **end if**
10: $t_n^{i,w} \leftarrow t_n^{i,w} + 1$
11: $\pi^{i+1} \leftarrow$ Optimize π^i with collected experiences and a DRL algorithm
12: $f_n^{i+1} \leftarrow f_n^i + \sum_{w=1}^{M} f_n^{i,w}$ for all n
13: $t_n^{i+1} \leftarrow t_n^i + \sum_{w=1}^{M} t_n^{i,w}$ for all n
14: $P^{i+1} \leftarrow$ Compute sampling probability by Eq. (2) based on f_n^{i+1} and t_n^{i+1}
15: $f_n^{i+1,w} \leftarrow 0$ for all n and w
16: $t_n^{i+1,w} \leftarrow 0$ for all n and w

Fig. 1. Prioritized environment configuration.

4. Environment

4.1 Overview

The environment we used to verify and analyze the proposed algorithm is a custom autonomous drone flight simulator developed based on Gazebo [21] and ROS [22]. Its primary purpose is to train a drone navigation agent with deep reinforcement learning to navigate from the starting position to the goal position without collision. For that purpose, we have created a map, as shown in Fig. 2. The map consists of six rooms, and each room has a door. The drone can only move between rooms through this door. When the drone hits a wall, it is considered a collision.



(a) (b) (c)

Fig. 2. Simulation map for drone navigation: (a) left rotated view, (b) top view, and (c) right rotated view).

The episode termination conditions are as follows: when the drone reaches the goal, when the drone hits the wall, or when the episode step count exceeds the maximum step size of an episode. We set the maximum step size to 1,000 steps. The observation space, action space, and reward function are as follows:

- Continuous observation space with depth camera image, distance between the drone and the goal, and angle difference between the drone's heading and the goal.
- Discrete action space with the size of 15, which is the combination of forward movement speed (*linear.x*) of 0, +1, and +2 m/s and yaw rate (*angular.z*) of -π/2, - π/4 , 0, π/4, and π/2.
- Reward function with +2,000 for reaching the goal, -1,000 for collision, -1,000 for exceeding the maximum step size of an episode, -3 for every step if *linear.x* is 0 for over ten consecutive steps, and -1 for every step. Notice that the major portion of the reward function is related to the termination conditions of the episode.

4.2 Coordinate Sets

Among the configurable environment configurations, we have chosen a coordinate set. The coordinate set comprises several coordinates, where each coordinate represents the drone's starting position and the goal position. The reasons for the choice of the coordinate set are mainly two folds. First, it is one of the necessary components that should be varied, as otherwise, the agent tends to memorize a particular route instead of learning how to avoid obstacles and reach the goal. Second, it is easy to create various levels of curriculums with similar difficulties. We will not evaluate a combination with curriculum learning that is out of the scope, but the proposed algorithm can accelerate learning a curriculum.

We have created two coordinate sets to evaluate PEC performance on both easy and challenging curriculums. Coordinate set 1 in Table 1 is easy: mainly coordinates without walls in the line of sight between the starting and goal positions. On the other hand, coordinate set 2, listed in Table 2, is challenging. It consists of coordinates with walls in between the starting and goal positions. If c_0 in Table 1 is configured, the drone spawns at (0, -2) and navigates to the goal placed at (5, -2) in Fig. 2. Fig. 3 presents six sample coordinates, consisting of two coordinates only included in Table 1 ("light green"), two coordinates included in Tables 1 and 2 ("green"), and two coordinates only included in Table 2 ("blue"). Note that the capitalized "C" means the coordinate set, and the lower case "c" means the coordinate, e.g., "C1_c6" means that the coordinate "c6" in the "Coordinate set 1" which is {(0, 5), (5, 5)}. We can see that coordinates in Table 2 are more challenging than those in Table 1.

Table 1. Coordinate set 1

Coordinate	Positions	Coordinate	Positions
c_0	{(0, -2), (5, -2)}	c_1	{(5, -2), (0, -2)}
c_2	{(-1, 0), (-1, 4)}	c_3	{(-1, 4), (-1, 0)}
c_4	{(-5, 0), (-5, 5)}	c_5	{(-5, 5), (-5, 0)}
c_6	{(0, 5), (5, 5)}	c_7	{(5, 5), (0, 5)}
c_8	{(3, 2), (3, 6)}	c_9	{(3, 6), (3, 2)}
c_{10}	{(-5, -5), (-8, -8)}	c_{11}	{(-8, -8), (-5, -5)}
c_{12}	{(-7, -8), (-1, -8)}	c_{13}	{(-1, -8), (-7, -8)}
c_{14}	{(3, -5), (3, -2)}	c_{15}	{(3, -5), (3, -2)}
c_{16}	{(0, -2), (-5, -2)}	c_{17}	{(-5, -2), (0, -2)}
c_{18}	{(-8, -2), (6, -2)}	c_{19}	{(6, -2), (-8, -2)}

Table 2. Coordinate set 2

Coordinate	Positions	Coordinate	Positions
c_0	{(-7, -8), (-1, -8)}	c_1	{(-1, -8), (-7, -8)}
c_2	{(-8, -2), (6, -2)}	c_3	{(6, -2), (-8, -2)}
c_4	{(-1, 3), (0, -5)}	c_5	{(0, -5), (-1, 3)}
c_6	{(0, 5), (5, -2)}	c_7	{(5,- 2), (0, 5)}
c_8	{(-1, -2), (5, 5)}	c_9	{(5, 5), (-1, -2)}
c_{10}	{(0, -1), (-5, 5)}	c_{11}	{(-5, 5), (0, -1)}
c_{12}	{(-6, -6), (-1, -8)}	c_{13}	{(-1, -8), (-6, -6)}
c_{14}	{(5, -7), (2, -1)}	c_{15}	{(2, -1), (5, -7)}
c_{16}	{(6, -1), (3, -6)}	c_{17}	{(3, -6), (6, -1)}
c_{18}	{(-7, -6), (-1, -8)}	c_{19}	{(-1, -8), (-7, -6)}
c_{20}	{(-5, 5), (0, -2)}	c_{21}	{(0, -2), (-5, 5)}
c_{22}	{(-7, 0), (5, 0)}	c_{23}	{(5, 0), (-7, 0)}

Fig. 3. Visualization of six sample coordinates in Tables 1 and 2.

5. Experiment Results

PEC is general enough to be applied to both on-policy and off-policy reinforcement learning algorithms. First, we present experiment results of PEC combined with PPO, which is an on-policy

algorithm, and then we show the results of PEC combined with Ape-X DQN, which is an off-policy algorithm.

5.1 Hardware and Software Configurations

We setup the experiment on Dell Precision 7920 with Xeon Gold 6240, 32 GB RAM, and a single Nvidia RTX 8000 48GB graphic card. Installed OS is Ubuntu 18.04. The simulator is based on Gazebo 9 and ROS Melodic. We implemented the algorithm based on Ray [23] version 0.8.5, which contains RLlib [24] with Ape-X DQN, and PPO implementations. TensorFlow [25] version 1.15.0 is utilized to execute these implementations. With the help of Ray, in the following experiments, 5 number of workers, i.e., simulators, in parallel are utilized to gather the experiences.

5.2 PEC-PPO with Coordinate Set 1

Hyperparameters of Ray, which are *num_gpus*, *num_workers*, *lambda*, *clip_param*, *kl_coeff*, *train_batch_size*, and *batch_mode*, are set to 1, 5, 0.95, 0.2, 1.0, 5000, and *complete_episode*, respectively. Other hyperparameters are set to default values. α in Equation (1) is set to 0.2. The training with the coordinate set 1 in Table 1 is performed for 1500 training iterations.

Fig. 4 is the average goal ratio over training iterations. We can observe that PEC's performance is slightly lower than w/o PEC's at the beginning of the training. PEC selects a coordinate for every episode by prioritizing the coordinates with high failure ratios. In other words, PEC is more likely to sample the coordinates with high failure ratios than w/o PEC which randomly samples from a uniform distribution. Therefore, when the overall failure ratio is high, especially in the early stages of training, training with PEC may seem slow. However, from the point of training iteration 140, when learning has been done to some extent, average goal ratios of PEC rise steeply and generally maintain high performance until the end of the training. As for the average goal ratio of the last 100 iterations, PEC is 0.959, and w/o PEC is 0.930, showing high performance in PEC. This is because PEC performs rollout through prioritized sampling for coordinates effective for training, and learns from the trajectories obtained through that rollout. On average, the time taken per training iteration is 42.369 seconds, of which the proposed algorithm occupies 0.681 seconds, only 1.607%.

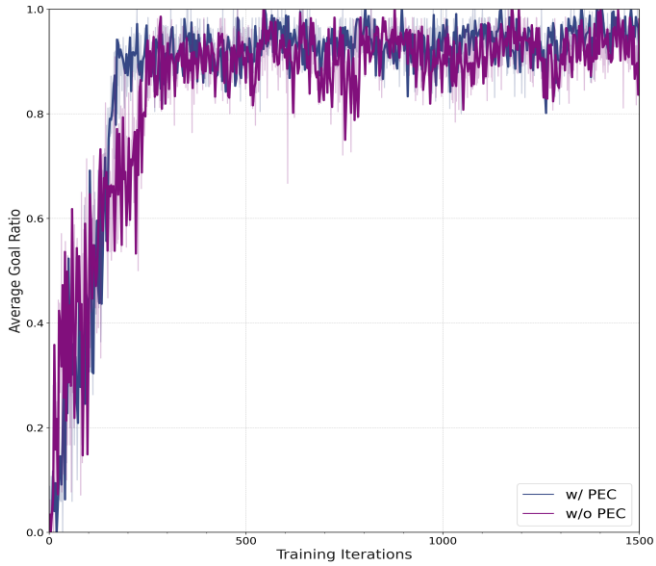


Fig. 4. Average goal ratio in training for PPO and coordinate set 1.

Fig. 5 is the trial ratio and failure ratio for each coordinate in 100th, 500th, 1000th, and 1500th training iterations. The upper row is with PEC, and the lower row is w/o PEC. The bar and the line represent the trial ratio and the failure ratio, respectively. Refer to Table 1 for the actual coordinates in x-axis. We can see that PEC effectively reduces the difficult coordinates' failure ratios and the overall failure ratios through the prioritized sampling. With PEC, the trial ratios are proportional to the failure ratios. The sampling gets more concentrated in difficult coordinates as the easy coordinates' failure ratios drop considerably compared to the difficult coordinates'. In particular, the trial ratio of c_{15}, the coordinate with the highest failure ratio, keeps increasing as the training iteration increases. It results in an effective decrease in the failure ratio of c_{15}. On the other hand, w/o PEC, the trial ratio is uniform regardless of the failure ratio. The failure ratio of c_{15} only slightly decreases as the training progresses. Furthermore, we can notice that PEC dynamically adjusts the trial ratios without the prior knowledge of the coordinates' difficulties, which usually requires manual settings by the experts, according to the failure ratios that dynamically changes as the training progresses.

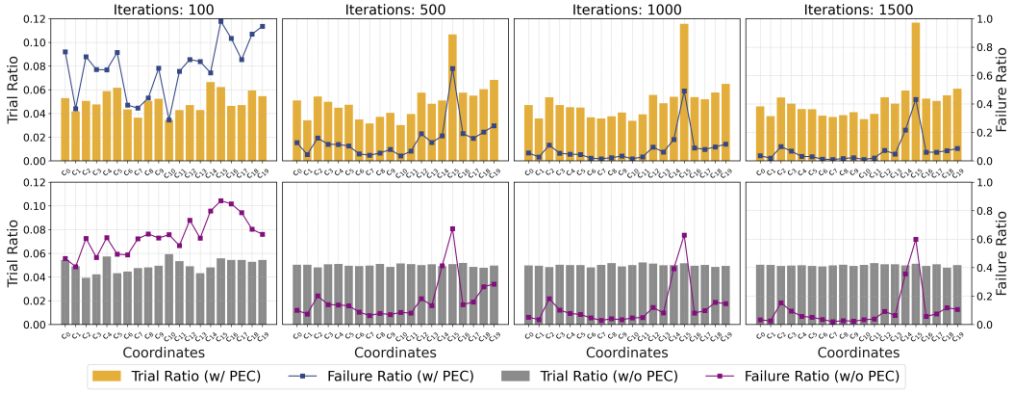


Fig. 5. Trial and failure ratio over coordinates for PPO and coordinate set 1.

The trends of trial and failure ratios over training iterations are presented in Fig. 6. Fig. 6(a) is the average and the standard deviation of the trial ratio, and Fig. 6(b) is the average and the standard deviation of the failure ratio over coordinates for every 100th training iteration. When PEC is applied, the standard deviation of the trial ratio is large. This is because prioritized sampling is performed according to the failure ratio with PEC: more samples with higher failure ratios and fewer samples with lower failure ratios. Without PEC, the standard deviations are small as it performs sampling from a uniform distribution. Moreover, as the number of samples increases as the training progresses, the standard deviation gradually decreases with the training iteration. The average trial ratios are all 0.05 because there are 20 coordination sets in the coordinate set 1. From Fig. 6(b), we can see that the standard deviation and average effectively decrease as training progresses with PEC, whereas standard deviation and average decrease but remain relatively high w/o PEC.

Fig. 7 is the average goal ratio and reward when evaluated 50 times using the checkpoint saved for every 100th training iteration. During the evaluation, random sampling from a uniform distribution was performed. PEC performs better than w/o PEC at all checkpoints. Through this, we can see that PEC is actually learning higher performance intelligence even in the early stage of training compared to w/o PEC. Although PEC may appear to have lower performance as the average goal ratio in the early stage of training is lower than w/o PEC. One more thing, as the major portions of the reward function are related to the episode termination conditions, average goal ratio and average reward graphs show similar tendency.

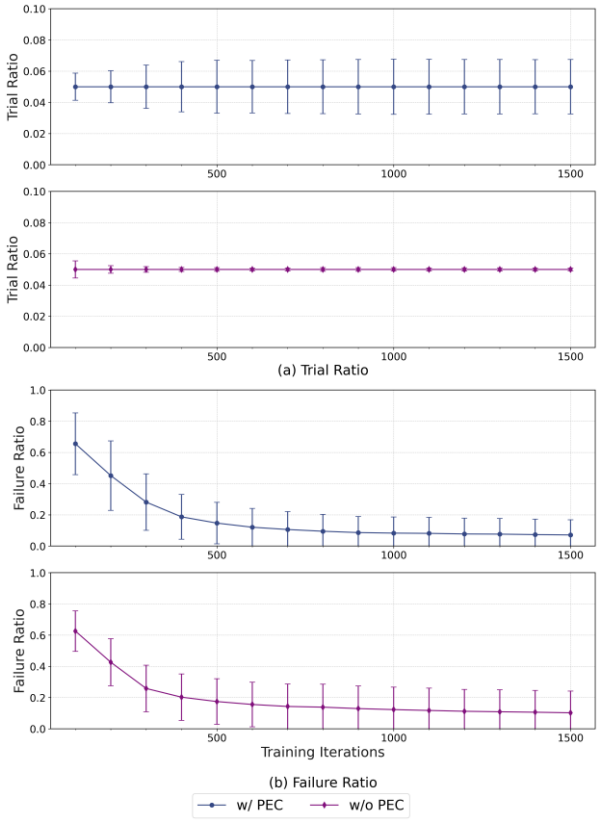


Fig. 6. Average and standard deviation of (a) trial ratio and (b) failure ratio over training iterations for PPO and coordinate set 1.

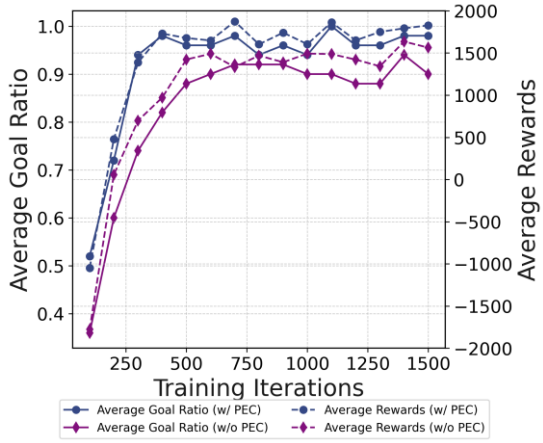


Fig. 7. Average goal ratio and reward in evaluation for PPO and coordinate set 1.

5.3 PEC-PPO with Coordinate Set 2

The same hyperparameters are used as in the previous subsection. The training with the coordinate set 2 in Table 2 is performed for 5000 training iterations.

The overall trend of Fig. 8 is similar to that of Fig. 4. However, we can clearly see the performance gap between PEC and w/o PEC. The last 100 iterations' average goal ratio is 0.860 and 0.764 for PEC

and w/o PEC, respectively. These results represent that the more difficult the task is, the greater the learning effectiveness through prioritized sampling with PEC. For coordinate set 2, which has four more environment configurations than coordinate set 1, the average time per training iteration is 41.461 seconds, of which the proposed algorithm occupies 0.692 seconds, only 1.669%.

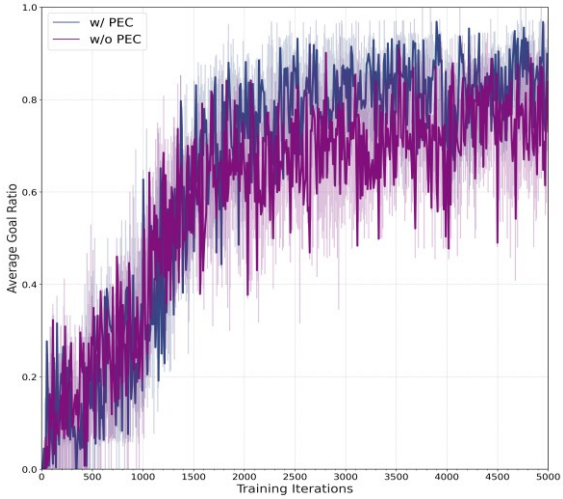


Fig. 8. Average goal ratio in training for PPO and coordinate set 2.

Fig. 9 also clearly presents the difference between PEC and w/o PEC. Even when there are many difficult coordinates, PEC effectively reduces the failure ratio by prioritizing the more difficult ones. With PEC, the minimum trial ratio continues to decrease and the maximum trial ratio continues to increase as the training progresses (the minimum values of iteration 1000, 2000, 3000, 4000, and 5000 are 0.0313, 0.0295, 0.0279, 0.0271, and 0.0269, respectively, and the maximum values of iteration 1000, 2000, 3000, 4000, and 5000 are 0.0470, 0.0505, 0.0553, 0.0593, and 0.0616, respectively). On the other hand, the trial ratio w/o PEC converges to the average resulting prolonged decrease of failure ratios.

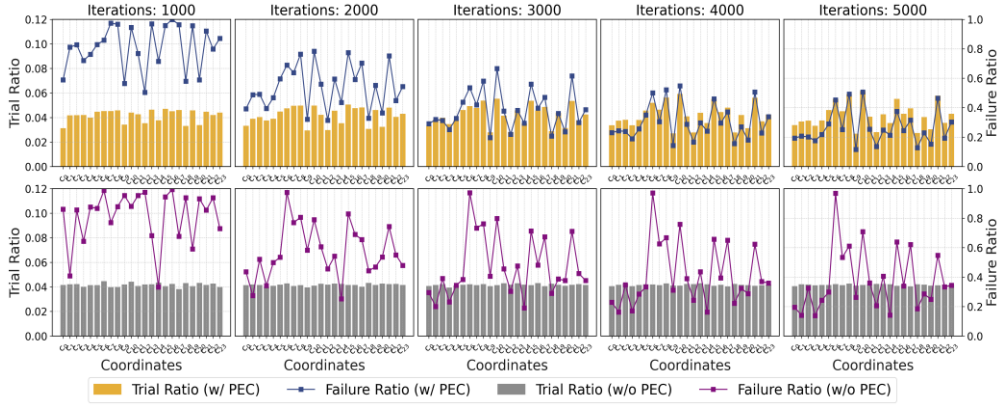


Fig. 9. Trial and failure ratio over coordinates for PPO and coordinate set 2.

From Fig. 10, it can be seen that the overall trend of standard deviation and average of trial and failure ratio over training iterations are similar to that of the coordinate set 1. However, the gap between w/ PEC and w/o PEC becomes noticeable. Since the coordinate set 2 is composed of 24 coordinates, the average trial ratio is 0.417.

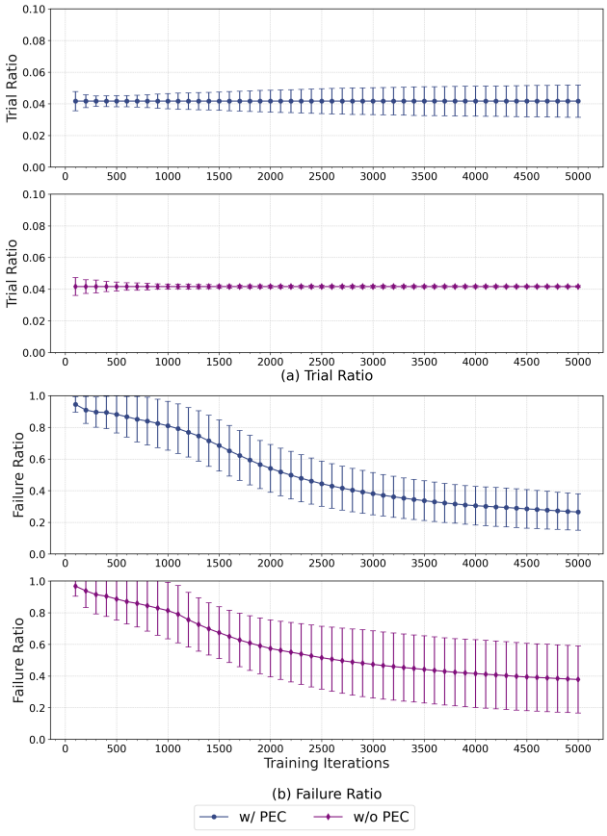


Fig. 10. Average and standard deviation of (a) trial ratio and (b) failure ratio over training iterations for PPO & coordinate set 2.

5.4 Analysis of α with PEC-PPO

All settings are the same as in Section 4.2, but only α is different which varies from 0 to 0.4.

Fig. 11 shows the results of monitoring the trial and failure ratios of every coordinate at the training iterations 100, 800, and 1500 with different α. The smaller the α, the higher the dependency to the failure ratio when prioritizing the coordinate sampling, and the higher the α, the lower it is. It yields the trade-off among the coordinates with higher and smaller failure ratios. The smaller the alpha, the more the trials are concentrated at the coordinates with the higher failure ratios, so the failure ratios effectively decrease. However, the failure ratios of other points decrease slowly. This phenomenon can be found in the figure.

When α is below a certain threshold, 0, 0.1, and 0.2 in Fig. 11, the standard deviation of trial ratio tends to increase, whereas it tends to decrease for α above a certain threshold, 0.3, and 0.4 in the figure. This is because the degree of dependency on the failure ratio in prioritization varies with α. Besides, the performance differs according to α due to the trade-off mentioned in Fig. 12. The performance with regard to the final average failure ratio is in the order of α 0.2, 0.0, 0.3, 0.1, and 0.4.

5.5 PEC-Apex with Coordinate Set 1

Hyperparameters of Ray, which are *num_gpus*, *num_workers*, *target_network_update_freq*, *gamma*, *train_batch_size*, and *batch_mode*, are set to 1, 5, 20000, 0.99, 5000, and *complete_episode*, respectively. Other hyperparameters are set to default values. α in Equation (1) is set to 0.2. The training with the coordinate set 1 in Table 1 is performed for 2000 training iterations.

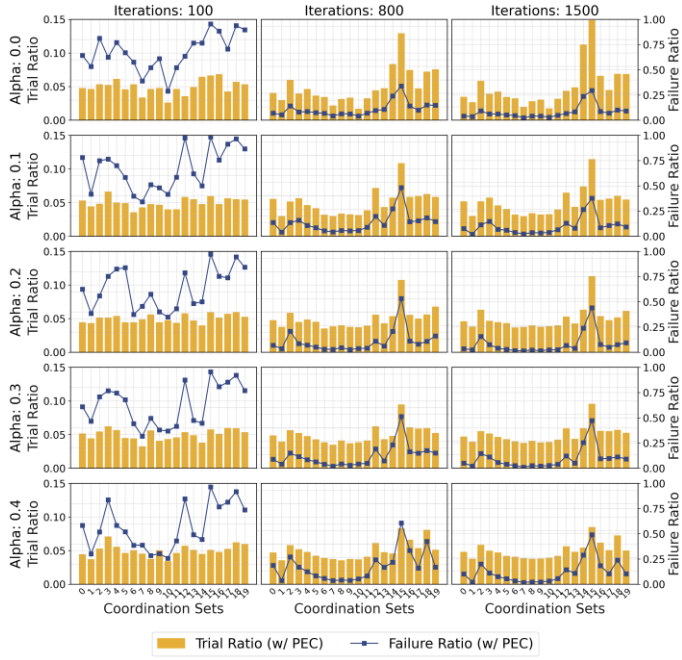


Fig. 11. Trial and failure ratio over coordinates with different α.

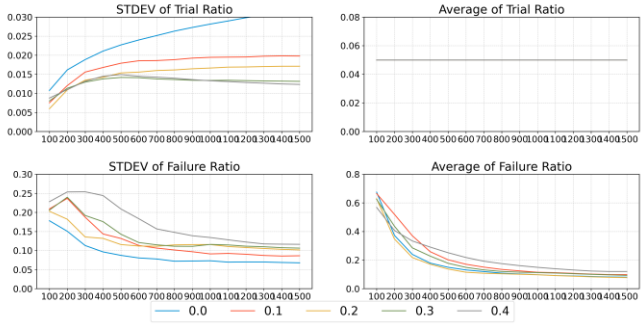


Fig. 12. Standard deviation and average of trial and failure ratio over training iterations with different α.

The overall trend is similar to Section 4.2. So, we are going to mention only worth to notice.

The interesting thing in Fig. 13 is that c_{18} shows the highest failure ratio unlike other results where c_{15} shows the highest failure ratio. We can assume that the difficulty felt by the agent may vary according to the early stage of the training process, and that even if it changes, PEC adapts well accordingly.

Fig. 14 is the average goal ratio over training iterations. This figure has another purpose, which is to empirically compare the training speed of w/ PEC and w/o PEC. We set the exit condition for the training process and presented the results. The exit condition is the average goal ratios of five consecutive iterations being maintained above a certain threshold. When the threshold is set to 0.9, displayed as *dash-dot*, the training with PEC, and w/o PEC terminates at 695th, and 1470th iteration, respectively, which implies that the training speed is increased by 52.7%. When the threshold is set to 0.95, displayed as *dot*, it takes 1342, and 1892 iterations to terminate the training with PEC and w/o PEC, respectively, resulting in 29.1% enhancement. The time it takes for a single training iteration is similar for PEC and w/o PEC. The total time taken for 2000 training iterations was 107, and 106 hours for PEC and w/o PEC, respectively. So, we just compared the number of training iterations for comparing the learning speed. From the above results, we can confirm that both the training speed and performance improve with PEC.

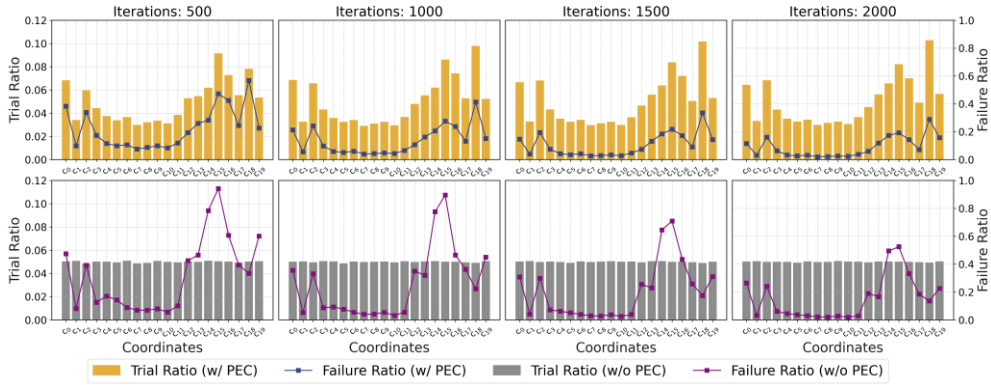


Fig. 13. Trial and failure ratio over coordinates for Apex-X DQN and coordinate set 1.

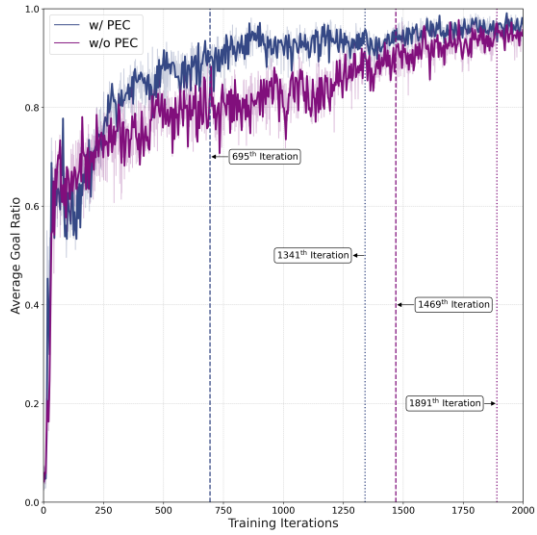


Fig. 14. Average goal ratio in training for Ape-X DQN and coordinate set 1.

6. Conclusion

This paper proposes a PEC. Examples of environment configurations are coordinate, map id, weather, and wind speed, which the simulator configures to initialize episodes. By prioritizing effective environment configurations and sampling accordingly, we can collect effect experiences for training the agents. We found out that PEC enhances both training speed and performance of reinforcement learning compared to the uniform sampling by applying it to the drone pathfinding. Moreover, we show that the proposed algorithm binds well to Ape-X DQN, an off-policy algorithm, and PPO, an on-policy algorithm. The same concept can be applied to a multi-task problem as the environment configuration can easily be extended to include task configuration, which is left for the future works of the paper.

Acknowledgements

Not applicable.

Author's Contributions

Conceptualization, SJ. Investigation and methodology, SJ, CC. Writing of the original draft, SJ.

Writing of the review and editing, CC. Software, SJ. Validation, SJ, CC. Project administration, CC. Funding acquisition, SJ. All the authors have proofread the final version.

Funding

This work was supported in part by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (No. 21ZR1100, A study of hyper-connected thinking Internet Technology by autonomous connecting, controlling and evolving ways), and in part by the research fund of Hanbat National University in 2021.

Competing Interests

The authors declare that they have no competing interests.

References

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2018, pp. 3207-3214.

[2] M. A. Rahman, Y. D. Lee, and I. Koo, "An efficient transmission mode selection based on reinforcement learning for cooperative cognitive radio networks," *Human-centric Computing and Information Sciences*, vol. 6, article no. 2, 2016. https://doi.org/10.1186/s13673-016-0057-2

[3] A. Belhadi, Y. Djenouri, G. Srivastava, and J. C. W. Lin, "Reinforcement learning multi-agent system for faults diagnosis of mircoservices in industrial settings," *Computer Communications*, vol. 177, pp. 213-219, 2021.

[4] U. Ahmed, J. C. W. Lin, and G. Srivastava, "Privacy-preserving deep reinforcement learning in vehicle adhoc networks," *IEEE Consumer Electronics Magazine*, 2021. https://doi.org/10.1109/MCE.2021.3088408

[5] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, et al., "Drone deep reinforcement learning: a review," *Electronics*, vol. 10, no. 9, article no. 999, 2021. https://doi.org/10.3390/electronics10090999

[6] V. J. Hodge, R. Hawkins, and R. Alexander, "Deep reinforcement learning for drone navigation using sensor data," *Neural Computing and Applications*, vol. 33, no. 6, pp. 2015-2033, 2021.

[7] B. Sharma, G. Srivastava, and J. C. W. Lin, "A bidirectional congestion control transport protocol for the internet of drones," *Computer Communications*, vol. 153, pp. 102-116, 2020.

[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016.

[9] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017 [Online]. Available: https://arxiv.org/abs/1707.06347.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 2018, pp. 1856-1865.

[12] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: a framework and survey," *Journal of Machine Learning Research*, vol. 21, article no. 181, 2020.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," 2017 [Online]. Available: https://arxiv.org/abs/1710.06537.

[15] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, et al., “Solving Rubik's cube with a robot hand,” 2019 [Online]. Available: https://arxiv.org/abs/1910.07113.

[16] S. Jang and M. Han, “Combining reward shaping and curriculum learning for training agents with high dimensional continuous action spaces,” in *Proceedings of 2018 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea, 2018, pp. 1391-1393.

[17] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in *Proceedings of the 35th International Conference on Machine Learning (ICML),* Stockholm, Sweden, 2018, pp. 1514-1523.

[18] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, “Teacher–student curriculum learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3732-3740, 2019.

[19] M. Wen, J. Park, and K. Cho, “A scenario generation pipeline for autonomous vehicle simulators,” *Human-centric Computing and Information Sciences*, vol. 10, article no. 24, 2020. https://doi.org/10.1186/s13673-020-00231-z

[20] K. Bringmann and K. Panagiotou, “Efficient sampling methods for discrete distributions,” *Algorithmica*, vol. 79, no. 2, pp. 484-508, 2017.

[21] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, Sendai, Japan, 2004, pp. 2149-2154.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System,” in *Proceedings of the ICRA Workshop on Open Source Software*, Kobe, Japan, 2009.

[23] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, et al., “Ray: a distributed framework for emerging AI applications,” in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Carlsbad, CA, 2018, pp. 561-577.

[24] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, et al., “RLlib: abstractions for distributed reinforcement learning,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholm, Sweden, 2018, pp. 3053-3062.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al., “TensorFlow: a system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265-283.