Safety Guided Policy Optimization

Dohyeong Kim[1], Yunho Kim[2], Kyungjae Lee[3], and Songhwai Oh[1]

***Abstract*— In reinforcement learning (RL), exploration is essential to achieve a globally optimal policy but unconstrained exploration can cause damages to robots and nearby people. To handle this safety issue in exploration, safe RL has been proposed to keep the agent under the specified safety constraints while maximizing cumulative rewards. This paper introduces a new safe RL method which can be applied to robots to operate under the safety constraints while learning. The key component of the proposed method is the *safeguard* module. The safeguard predicts the constraints in the near future and corrects actions such that the predicted constraints are not violated. Since actions are safely modified by the safeguard during exploration and policies are trained to imitate the corrected actions, the agent can safely explore. Additionally, the safeguard is sample efficient as it does not require long horizontal trajectories for training, so constraints can be satisfied within short time steps. The proposed method is extensively evaluated in simulation and experiments using a real robot. The results show that the proposed method achieves the best performance while satisfying safety constraints with minimal interaction with environments in all experiments.**

I. INTRODUCTION

Reinforcement learning (RL) has shown excellent results in robot navigation, manipulation, locomotion, and other robot tasks [1]–[6]. However, to train robots using RL in the real world beyond simulation, safety becomes the most important issue to be addressed first. An RL agent needs to explore for better performance but the same exploration action may cause damages to the robot and its surroundings, as well as people nearby. For example, a robot might fall off the floor, which causes repair costs, or sudden acceleration may cause injuries to nearby people. To deal with these problems, it is necessary to develop a safe exploration method that satisfies constraints such as keeping a distance from obstacles. Additionally, constraint violations inevitably occur in early training stages due to a lack of information about environments, so we focus on how to solve RL problems while satisfying constraints as quickly as possible.

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01190, [SW Star Lab] Robot Learning: Efficient, Safe, and Socially-Acceptable Machine Learning, 40%), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2022R1A2C2008239, General-Purpose Deep Reinforcement Learning Using Metaverse for Real World Applications, 30%), and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00331, Development of emotion recognition/generation-based interacting edge device technology for mental health care, 30%). *(Corresponding authors: Songhwai Oh.)*

[1]D. Kim and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul 08826, Korea (e-mail: dohyeong.kim@rllab.snu.ac.kr, songhwai@snu.ac.kr).

[2]Y. Kim is with the Robotics and Artificial Intelligence Lab, Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology, Yuseong-gu, Daejeon 34141, Republic of Korea (e-mail: awesomericky@kaist.ac.kr).

[3]K. Lee is with the Department of Artificial Intelligence, Chung-Ang University, Seoul 06974, Korea (e-mail: dlxhrl@cau.ac.kr).

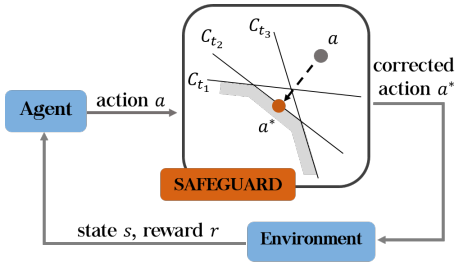


Fig. 1: An overview of safety guided policy optimization (SGPO) with a safeguard module. An action sampled from a learning agent is corrected by the safeguard module (Algorithm 1) to satisfy sequential future step-wise constraints. The constraints are linearly approximated to an action and indicated by C_{t_1}, C_{t_2}, and C_{t_3}, and the area that satisfies all constraints is indicated in gray. After receiving the corrected action, the environment returns a new state and a reward.

A Markov decision process (MDP) with an additional set of cost functions is called a constrained Markov decision process (CMDP) [7]. RL problems with safety constraints, called *safe RL*, are usually defined under the CMDP framework. Safe RL problems can be solved using various policy update rules, denoted by *safe policy updates*, such as Lagrangian methods [8]–[10] or trust-region methods [11]. For safe exploration, safety constraints defined in CMDPs should be uniformly bounded over all time steps, but safe policy updates are difficult to guarantee the uniform boundness property. Thus, additional devices such as shielding methods [12] are needed to quickly learn a policy with uniformly bounded safety constraints. Shielding methods consist of a safety monitoring module which determines whether a given state and action pair is safe and a safety reflection module which modifies the action to satisfy safety constraints [12]. In Section III-B, we show that a shielding method makes a policy to have uniformly bounded safety constraints during the overall training process via a toy example.

Shielding-based safe RL methods typically use safety critics for the safety monitoring module [13], [14]. These critic-based methods judge safety through the safety critic value for a given state-action pair, and the Bellman equation is used to update the safety critic with sampled transitions from the current policy. However, there are two limitations on the safety critic. First, to learn safety critics with low bias, several long-time horizon trajectories have to be sampled for a given state-action, which is not sample-efficient. Second, additional policies are required to calculate a safe action when the action from the current policy is judged unsafe. To avoid using additional policies, Bharadhwaj et al. [13] have proposed a method of repeatedly sampling an action until it finds a safe one, but there is no upper bound on the number of required samples, and the action calculation takes

longer as the number of samples increases.

In this paper, we propose *safety guided policy optimization* (SGPO), a shielding-based safe RL method, which allows an RL agent to satisfy constraints with minimal interactions with environments. To overcome the shortcomings of the critic-based shielding methods, a prediction-based shielding method, called *safeguard*, is proposed, and the overview is shown in Figure 1. The safeguard predicts sequential short-term future costs for the safety monitoring module, resulting in higher sample efficiency than the critic-based shielding methods. Furthermore, the safeguard linearizes the predicted sequential costs to the action and use them to correct the action safely, so additional policies are not needed, which is motivated by Dalal [15]. The proposed method, SGPO, not only uses the safeguard to execute safe actions during exploration but also uses constrained policy optimization (CPO) [11] to perform safe policy updates. To this end, we divide the safety constraints into step-wise constraints used for the safeguard and trajectory-wise constraints used for the CPO policy updates. To give an example of dividing the constraints, for a speed limit case, a step-wise constraint can be set to limit the instantaneous speed and a trajectory-wise constraint can be set to limit the number of the speed violations during an episode below one. Then, feasible policy sets for both constraints are the same set of policies that drive below the speed limit at every step. Additionally, the objective function of CPO is modified with an introduction of a term to make policies to imitate the outputs of the safeguard, which means the safeguard serves as a guide.

The proposed method has been extensively evaluated in simulation and sim-to-real experiments. In simulation, *Car Goal*, *Point Goal*, and *Doggo Goal* tasks provided by the Safety Gym [16] (Figure 4) are used. For the sim-to-real experiments, a navigation task without bumping into any obstacles using a Jackal robot from Clearpath [17] is used. The proposed SGPO is compared with various RL methods, including another shielding-based safe RL method. The simulation and the sim-to-real experimental results show that the proposed method satisfies constraints with the least interactions, and has the highest scores for all tasks.

II. RELATED WORK

There are a number of RL methods for safe exploration, and they can be divided into two categories according to how constraints are handled. First, there are methods to limit actions through safety monitoring, which we refer to as shielding [12], and secondly, there are methods that reflect constraints when update policies.

1) Shielding Based Safe RL: Methods of using safety critics for safety monitoring are proposed in [13], [14]. A method proposed by in [14], called recovery-RL (RRL), uses a discounted sum of probabilities of constraint violation as safety critics and trains not only a policy that maximizes the reward sum, but also a recovery policy that learns the safety critics to get a safe action. However, to train the safety critics with low bias, several samples of long time horizons are needed, so the accuracy of the safety monitoring is imperfect in the early phase of training. Ohnishi et al. [18] and Cheng et al. [19] proposed methods that learn transition models and determine whether a state predicted by the model is safe using a control barrier function. The method proposed in [18] is proved that the agent can stay or return to a safe state using the Lyapunov stability, but it is difficult to find a function that satisfies the condition to be a control barrier function. A method proposed by Dalal et al. [15] defines a structure called a *safety layer* that predicts costs after one step. In the safety layer, the costs are linearly approximated to the action, and unsafe actions are corrected subject to the linearized constraints. Due to this process, the safety layer has the advantage of performing both safety monitoring and reflection. However, since the safety layer predicts costs after only one step, it is difficult to apply when several steps are needed to be safe.

2) Update Based Safe RL: Chow et al. [10] have converted a constrained RL problem to an unconstrained problem with Lagrange multipliers and proposed a primal-dual method that updates policies and the multipliers concurrently. Achiam et al. [11] have proposed a trust region method, called CPO. Surrogate constraints are obtained within the trust region, and a policy is updated by solving a Lagrangian dual problem of the original problem with the surrogate constraints. A method proposed by Zhu et al. [20] trains an actor policy to maximize cumulative reward and another advisor policy to minimize cumulative cost. The agent explores with a policy obtained by the affine combination of two policies, and the affine weight is gradually shifted to the actor from the advisor. Although there are various safe policy update methods, the proposed method uses the CPO framework for policy updates because of the monotonous performance improvement and fewer parameters to be tuned.

There are other safe RL methods. In [21], Lyapunov functions are explicitly formulated using auxiliary costs, and policies are updated in the Lyapunov-induced policy set [21]. In the method proposed by Saunders et al. [22], a person corrects actions when agents try to perform unsafe actions, and an imitator is trained to imitate behaviors of the person through supervised learning manners. After trained, the imitator substitutes the role of the person for agents to explore safely without human interactions.

III. BACKGROUND

A. Constrained Markov Decision Process

A constrained Markov decision process (CMDP) is a variant of an MDP, which adds a set of cost functions. A CMDP is expressed as a tuple $(\mathcal{S}, \mathcal{A}, \rho, \mathcal{P}, r, \gamma, \mathcal{C}, \mathcal{D})$, with a state space $\mathcal{S} \subset \mathbb{R}^n$, an action space $\mathcal{A} \subset \mathbb{R}^m$, an initial state distribution $\rho : \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$, a transition model $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$, a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, a discount factor $\gamma \in [0, 1)$, a cost function $c : \mathcal{S} \mapsto \mathbb{R}$, and a cost limit value $d \in \mathbb{R}$. A policy $\pi(a|s)$ represents an action distribution on a given state s and the discounted state distribution is denoted by $d_\pi(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathrm{Prob}(s_t = s|\rho, \pi, \mathcal{P})$. The value, action-value, and advantage functions are defined as follows:

$$
\begin{aligned}
V^\pi(s) &:= \mathop{\mathbb{E}}_{\pi,\mathcal{P}}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s], \\
Q^\pi(s,a) &:= \mathop{\mathbb{E}}_{\pi,\mathcal{P}}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a], \\
A^\pi(s,a) &:= Q^\pi(s,a) - V^\pi(s).
\end{aligned}
\tag{1}
$$

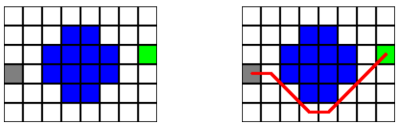
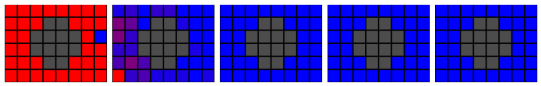
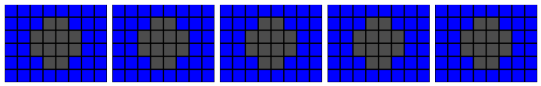


Fig. 2: The toy environment and an optimal path. An agent is spawned in a gray area, a target location is shown in green, and hazard zones are shown in blue.



(a) Without shielding



(b) With shielding

Fig. 3: Visualization of the true cost values. The true cost values are calculated by averaging the discounted cost sum obtained by performing 100 episodes starting at each grid. From the left to the right, evaluation and improvement are performed once. When the value is greater than or equal to 0.1, the grid is displayed in red, and when the value is closer to 0, the grid is displayed in blue.

The cost value $V_C^\pi(s)$, cost action-value $Q_C^\pi(s,a)$, and cost advantage functions $A_C^\pi(s,a)$ are defined by putting the cost function $c(s)$ instead of the reward function $r(s,a)$ in (1). From the CMDP setting, a safety constraint can be defined as $\mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}[\sum_{t=0}^{\infty}\gamma^t c(s_t)] \leq d$. Then, a safe RL problem can be formulated as follows:

$$\max_\pi \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] \quad \mathbf{s.t.} \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^t c(s_t)\right] \leq d, \quad (2)$$

where the expectation of the discounted cost sum is denoted by $J_C(\pi) = \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}[\sum_{t=0}^{\infty}\gamma^t c(s_t)]$.

B. Shielding Method

Shielding methods are used for safe exploration as demonstrated by Alshiekh et al. [12]. Shielding methods are divided into a *preemptive shielding* and a *post-posed shielding* in [12]. The preemptive shielding computes a set of safe actions for a given state and provides the set to the agent to select an action, and the post-posed shielding converts an action received from the agent into a safe action if the received action is not safe. We use the post-posed shielding method to construct the safeguard module as in Figure 1.

To show how the shielding methods perform safe exploration well, we build a toy example shown in Figure 2. Each state has one of 13 discrete velocities for each grid, and the agent moves with 5 acceleration actions under a speed limit. When the agent passes through the hazard zones, the agent gets a cost $+1$, and when the target location is reached, the agent gets a reward $+1$. Policy improvement and policy evaluation are alternated to get an optimal safe policy. Specifically, the shielding method limits the policy improvement to select an action from a set of actions whose expected costs of two future steps are less than 0.5. The results with and without the shielding method are shown in Figure 3. Even though the shielding method is not used, the cost values decrease as the update is performed, but the cost values of the shielding method are kept low throughout training. As a result, the toy example shows that the shielding method helps an agent keep the safety constraint uniformly bounded in all time steps.

IV. PROPOSED METHOD

The proposed method solves the safe RL problem (2) by dividing the safety constraints into step-wise constraints and trajectory-wise constraints. To this end, we introduce a new step cost function $c_{\text{step}} : \mathcal{S} \mapsto \mathbb{R}$ and reformulate the original cost function $c(s)$ as follows:

$$c(s) = \text{sigmoid}(K \cdot c_{\text{step}}(s)), \quad (3)$$

where K is a constant, and the cost function $c(s)$ becomes an indicator function if K is infinity. As sparse cost functions such as indicator functions make training the cost value functions difficult, we use a finite value for K. Then, the safe RL problem can be modified with step-wise constraints as follows:

$$\begin{aligned} &\underset{\pi}{\text{maximize}} \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] \\ &\mathbf{s.t.} \quad \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^t c(s_t)\right] \leq d, \\ &\text{Prob}\left(c_{\text{step}}(s_t) > 0 \middle| s_0 \sim \rho,\pi,\mathcal{P}\right) \leq \epsilon, \ \forall t, \end{aligned} \quad (4)$$

where ϵ is a threshold of the likelihood of the constraint violation. To handle the step-wise constraints, we separate the problem (4) into two sub-problems. The first sub-problem is to construct a shielding method, called *safeguard*, which corrects actions sampled from a policy to satisfy the step-wise constraints. In the safeguard module, actions are projected onto an action set that is the intersection of the step-wise constraints which are predicted by neural networks, called *look-ahead cost networks*. As the safeguard transforms actions and passes them to environments, it causes an effect of modifying the transition model from $\mathcal{P}$ to $\mathcal{P}_{\text{safe}}$ and the reward function from r to r_{safe}, which will be explained in detail in Section IV-B. The second sub-problem is to update policies to optimize the problem (4) of the CMDP with the modified transition model $\mathcal{P}_{\text{safe}}$ and reward function r_{safe}. Since the step-wise constraints are satisfied by the safeguard, the safe RL problem with step-wise constraints (4) can be solved using the following optimization problem:

$$\max_\pi \mathop{\mathbb{E}}_{\substack{\rho,\pi,\\ \mathcal{P}_{\text{safe}}}}\left[\sum_{t=0}^{\infty}\gamma^t r_{\text{safe}}(s_t,a_t)\right] \mathbf{s.t.} \mathop{\mathbb{E}}_{\substack{\rho,\pi,\\ \mathcal{P}_{\text{safe}}}}\left[\sum_{t=0}^{\infty}\gamma^t c(s_t)\right] \leq d. \quad (5)$$

The proposed method utilizes the safeguard to satisfy step-wise constraints and updates policies by solving the constrained optimization problem (5). The remainder of this section describes how to construct the safeguard module, the policy update rule, and the look-ahead cost networks.

A. Constructing Safeguard

The safeguard is introduced to handle the step-wise constraints (4) during exploration. To this end, the safeguard predicts sequential future costs for a given state-action pair and determines whether the step-wise constraints are satisfied. If an action violates any of step-wise the constraints,

Algorithm 1 Safeguard

Input: state s, action a
Output: action a^*

function $\mathrm{SG}(s, a)$
  Initialize $\mathcal{M}' = \mathcal{M}$.
  for $j = |\mathcal{M}|, |\mathcal{M}| - 1, ..., 1$ **do**
    $A_{\text{feasible}} = \{\tilde{a} | g_\pi^i(s;\theta_g^i)^T \tilde{a} + b_\pi^i(s;\theta_b^i) + c_{\text{step}}(s) \le d_{\text{step}},\ \forall i \in \mathcal{M}'\}$.
    if $A_{\text{feasible}} \neq \emptyset$ **then**
      Get $a^* = \underset{\tilde{a} \in A_{\text{feasible}}}{\operatorname{argmin}} ||\tilde{a} - a||^2$ using a QP solver.
      break
    end if
    Remove the jth component of $\mathcal{M}'$.
  end for
  return a^*
end function

it is safely corrected by projecting onto the intersection of the constraints. In this correcting process, the constraints are linearized to the action, and a safe action is calculated using a QP solver.

We first simplify the step-wise constraints (4). With the assumption that the step costs follow Gaussian distribution with fixed standard deviation σ, the step-wise constraints can be rewritten as follows:

$$\mathbb{E}\left[c_{\text{step}}(s_t) \middle| s_0 \sim \rho, \pi, \mathcal{P}\right] \le d_{\text{step}},\ \forall t, \tag{6}$$

where $d_{\text{step}} = -\sigma \cdot \mathrm{CDF}^{-1}(1-\epsilon)$ and CDF^{-1} is the inverse cumulative density function of the standard normal distribution. To predict sequential future step costs, a i step look-ahead cost function is defined and linearly-approximated to actions as follows:

$$\begin{aligned}\Delta C_\pi^i(s,a) &:= \mathop{\mathbb{E}}_{\pi,\mathcal{P}}[c_{\text{step}}(s_i) - c_{\text{step}}(s_0) | s_0 = s, a_0 = a] \\ &\approx g_\pi^i(s;\theta_g^i)^T a + b_\pi^i(s;\theta_b^i),\end{aligned} \tag{7}$$

where $g_\pi^i : \mathcal{S} \mapsto \mathbb{R}^m$ and $b_\pi^i : \mathcal{S} \mapsto \mathbb{R}$ are i step look-ahead cost networks, and θ_g^i and θ_b^i are parameters of the networks. The look-ahead cost function ΔC_π^i outputs the difference between the cost of the current state and the future state following the policy π during i time steps. Given the current state s and sampled action $a \sim \pi(\cdot|s)$, the action correcting process in the safeguard module can be formulated as follows with the look-ahead cost functions:

$$\begin{aligned}&a^* = \underset{\tilde{a}}{\operatorname{argmin}} ||\tilde{a} - a||^2 \\ &\mathbf{s.t.}\ \Delta g_\pi^i(s;\theta_g^i)^T a + b_\pi^i(s;\theta_b^i) + c_{\text{step}}(s) \le d_{\text{step}},\ \forall i \in \mathcal{M},\end{aligned} \tag{8}$$

where the corrected action a^* can be obtained by a QP solver [23]. Since treating all constraints for every time step is intractable, we only deal with constraints for time steps in a finite set $\mathcal{M}$ ($= \{1, 6, 11, 16, 21\}$ in experiments). However, it is possible that there is no feasible solution when the cost predicting networks are undertrained. Hence, we prioritize the constraints in the order of the nearest time step. When the feasible set is empty, we find the corrected action a^* by removing the constraint of the largest time step sequentially until there is a feasible solution. Using this process, the solution from the safeguard can satisfy the maximal set of constraints from the nearest time step. The overall process of the safeguard is shown in Algorithm 1.

B. Policy Update

With the safeguard module, we can consider modified transition $\mathcal{P}_{\text{safe}}$ and reward function r_{safe} for solving the original CMDP problem, where

$$\begin{aligned}\mathcal{P}_{\text{safe}}(s'|s,a) &= \mathcal{P}(s'|s,a^*), \\ r_{\text{safe}}(s,a) &= r(s,a^*),\end{aligned} \tag{9}$$

and $a^* = \mathrm{SG}(s,a)$, which is defined in Algorithm 1. The discounted state distribution of the modified CMDP is defined as $d_\pi^{\text{safe}}(s) = (1-\gamma)\sum_{t=0}^\infty \gamma^t \mathrm{Prob}(s_t = s | \rho, \pi, \mathcal{P}_{\text{safe}})$.

Policies explore environments in the modified CMDP and are updated to optimize (5). Since (5) is exactly the same problem of CPO, policies can be updated using the CPO algorithm. However, if $a \sim \pi(\cdot|s)$ and the corrected action a^* from Algorithm 1 are excessively different, the accuracy of the look-ahead cost functions may decrease and cause constraint violations. Therefore, the following loss is added to the objective function of CPO to increase the likelihood that the action sampled by the policy is identical to the action calculated from the safeguard.

$$\begin{aligned}L_{\text{guide}}(\pi) &:= \mathop{\mathbb{E}}_{s \sim d_{\pi_k}^{\text{safe}}}[D_{\text{KL}}(\mathrm{SG}(s, \pi_k(\cdot|s)) || \pi(\cdot|s))] \\ &= \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_k}^{\text{safe}} \\ a \sim \pi_k}}[-\log \pi(a^*|s) | a^* = \mathrm{SG}(s,a)],\end{aligned} \tag{10}$$

where $\mathrm{SG}(s, \pi_k(\cdot|s))$ is the result of transforming the action distribution of $\pi_k(\cdot|s)$ by the safeguard. Since the loss acts as a guide to the policy, it is called a *guide loss*. Then, the policy update problem can be written as follows:

$$\begin{aligned}&\pi_{k+1} = \underset{\pi}{\operatorname{argmax}} \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_k}^{\text{safe}} \\ a \sim \pi_k}} \left[\frac{\pi(a|s)}{\pi_k(a|s)} A^{\pi_k}(s,a)\right] - \eta L_{\text{guide}}(\pi) \\ &\mathbf{s.t.}\quad J_C(\pi_k) + \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_k}^{\text{safe}} \\ a \sim \pi_k}} \left[\frac{\pi(a|s)}{\pi_k(a|s)} A_C^{\pi_k}(s,a)\right] \le d, \\ &\qquad \mathop{\mathbb{E}}_{s \sim d_{\pi_k}^{\text{safe}}}[D_{\text{KL}}(\pi_k(\cdot|s) \| \pi(\cdot|s))] \le \delta,\end{aligned} \tag{11}$$

where η is a coefficient, and ϵ is a size of the trust region.

C. Look-Ahead Cost Network Update

In the proposed method, the look-ahead cost functions $\Delta C_\pi^i(s,a)$ are linearly approximated to actions using the look-ahead cost networks g_π^i and b_π^i, which can be updated in a supervised learning manner. Using (7), the approximated i step look-ahead cost is written as $g_\pi^i(s;\theta_g^i)^T a + b_\pi^i(s;\theta_b^i) + c_{\text{step}}(s)$. Then, after trajectories $\tau = (s_0, a_0, s_1, a_1, \dots)$ are collected, g_π^i and b_π^i are updated by minimizing the following loss.

$$\begin{aligned}L(\theta_g, \theta_b) := \sum_{i \in \mathcal{M}} \mathop{\mathbb{E}}_{\rho,\pi,\mathcal{P}} \Big[\Big| &g_\pi^i(s_t;\theta_g^i)^T a_t + b_\pi^i(s_t;\theta_b^i) \\ &+ c_{\text{step}}(s_t) - c_{\text{step}}(s_{t+i}) \Big| \Big].\end{aligned} \tag{12}$$

The overall SGPO algorithm is described in Algorithm 2.

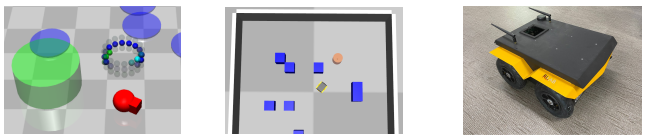
V. EXPERIMENTS

The proposed method, SGPO, is compared with other safe RL methods, including shielding-based safe RL methods. Additionally, to investigate how many look-ahead cost networks are needed for the safeguard, we experiment with

Algorithm 2 Safety Guided Policy Optimization

Input: Initial policy networks $\pi(a|s;\phi)$, initial look-ahead cost networks $g^i_\pi(s;\theta^i_g)$ and $b^i_\pi(s;\theta^i_b)$ for $i \in \mathcal{M}$.

for epochs=1, P **do**
  Initialize trajectory memory D.
  for episodes=1, E **do**
    Get an initial state s_0 from the environment.
    for t=0, T **do**
      Sample an action $a_t \sim \pi(\cdot|s_t;\phi)$ and get the corrected action a^*_t from Algorithm 1.
      Feed the action a^*_t to the environment.
      Get reward r_t, cost c_t, and next state s_{t+1}, and store $(s_t, a_t, a^*_t, r_t, c_t, s_{t+1})$ in D.
    end for
  end for
  Update $\pi(a|s;\phi)$ with D by minimizing (11) using the CPO method.
  Update $g^i_\pi(s;\theta^i_g)$ and $b^i_\pi(s;\theta^i_b)$ for $i \in \mathcal{M}$ with D as described in (12).
end for



(a) Safety Gym (b) Jackal Sim (c) Jackal Robot

Fig. 4: (a) shows the Safety Gym, and (b) and (c) show the experimental environment for the Jackal robot.

different sets of $\mathcal{M}$ in Section V-D. In the rest of this section, the simulation and the sim-to-real experimental setup are described, and the experimental results are analyzed.

A. Simulation Setup

For simulation, *Point Goal*, *Car Goal*, and *Doggo Goal* tasks from the Safety Gym [16] are used. These tasks are to make robots reach the goal as quickly as possible without passing through hazard regions as shown in Figure 4a.

The state space for each task is a 24-dimensional space which consists of an accelerometer, a velocimeter, and a gyrometer, and LiDAR data. The dimension of action space for each task is two. Since *Doggo Goal* is a complex task to learn, the environment is hierarchically reconstructed. First, a low-level policy of navigating to a goal without obstacles is trained using TRPO [24], and policies of safe RL methods explore the environment by giving two-dimensional actions to the low-level policy as a goal position. The number of the hazard is eight, and if the agent gets into hazard regions, the number of constraint violations (CV) increases by one.

The reward function used in the experiment is followings:

$$r(s,a) = \Delta d_g + \begin{cases} 1, & \text{if } d_g \leq \delta \\ 0, & \text{otherwise} \end{cases}, \tag{13}$$

where d_g is distance from the agent to the goal, Δd_g is the reduction of the distance to the goal from last time step, and δ is the goal size ($0.3\,m$ in experiments). The step cost function is defined as $c_{\text{step}}(s) = r_h - d_h$, where r_h is a radius of the hazard area (0.2 in experiments), and d_h is the minimum distance to obstacles from LiDAR. The cost function c, which is used for the trajectory-wise constraint, is computed using (3) with a constant K (10 in experiments).

B. Sim-to-Real Experiment Setup

For the sim-to-real experiment, the Jackal robot from Clearpath [17] shown in Figure 4c is used. The task is to navigate to goals without bumping into obstacles or vertical walls shown in Figure 4b. The state space is 31 dimensions with LiDAR data, a relative goal pose, and speed values. The action consists of two-dimensional values that control steering and linear acceleration, and the robot can only move forward as the LiDAR does not cover the back. The reward and the cost function are the same as the simulation setup. However, unlike the Safety Gym tasks, episodes can be terminated early if the measured distance between the partition and the robot is smaller than $30\,cm$ for safety.

For all tasks, five step-wise constraints are used ($\mathcal{M} = \{1, 6, 11, 16, 21\}$). The limit of the step-wise constraints d_{step} is -0.1 for the simulation experiments and -0.2 for the sim-to-real experiments. The limit of the trajectory-wise constraint d is 0.025 and η for the guide loss is 0.1.

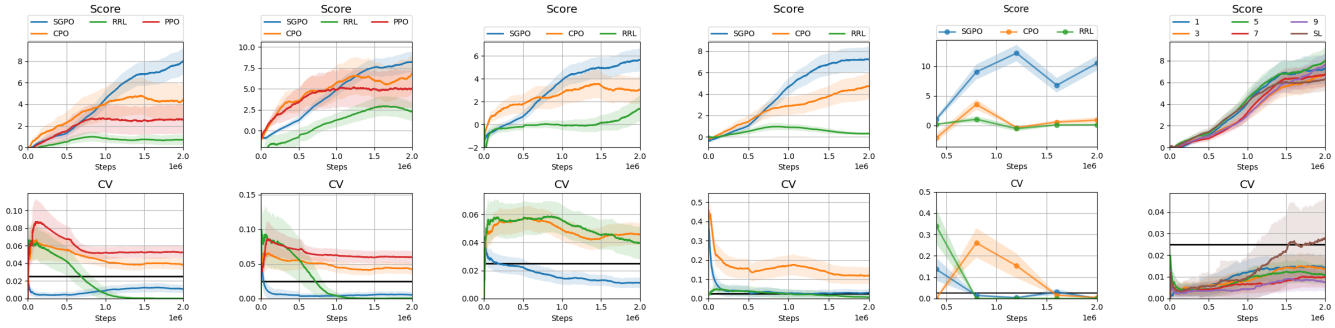
C. Results and Analysis

PPO [25], an RL algorithm, CPO [11], a safe RL algorithm, and RRL [14], another shielding-based algorithm, are used as baseline methods for performance comparison. The following score metric is used to measure the performance.

$$\text{score} = \sum_{t=0}^{T} r_t \Big/ \left(1 + \sum_{t=0}^{T} \text{CV}_t\right). \tag{14}$$

The metric (14) means that the score is penalized by diluting the reward sum as the number of constraint violation increases. The results of the simulation and the Jackal experiment are shown in Figure 5. In the case of the Jackal task, for every 400,000 steps, 10 episodes are performed in the real environment for evaluation. The proposed method, SGPO, is indicated by blue lines, and CPO, RRL, and PPO are indicated by orange, green, and red, respectively. The upper graphs in Figure 5 show the scores over the training steps, the lower graphs show the average costs of the trajectory-wise constraint per episode, and the limit value for the constraint is indicated by a black solid line.

From the results shown in Figure 5, SGPO shows the highest scores in all tasks. Also, SGPO satisfies the constraints in 0 steps, 30,000 steps, 120,000 steps, and 500,000 steps in the point, car, doggo, and jackal, respectively, which is the fewest number of interactions in all the tasks. In the real environment, the constraints are satisfied after the first 400,000 steps, and the score is also the highest, showing that SGPO is suitable for real-world application. From the fact that scores of PPO are low and the average costs are not reduced, it can be inferred that safe RL methods are required to learn the given tasks. RRL satisfies the constraints after 730,000 steps in the point and car tasks, 750,000 steps in the Jackal, and do not satisfy the constraint in the doggo task. This result shows that training the safety-critic is not sample-efficient. CPO shows the second-best performance after SGPO in all tasks, but it does not satisfy any constraints. From this, it can be seen that it is difficult to perform safe exploration during training without a shielding method. In the Jackal experiment, CPO shows similar performance to SGPO, but the constraint violation is not low enough, so it is difficult to apply to real robots.



(a) Point Goal (b) Car Goal (c) Doggo Goal (d) Jackal Sim (e) Real-World (f) Ablation Study

Fig. 5: Training curves of simulations, jackal experiments, and ablation studies. In each figure, upper graph is score during training steps, and lower one is th average number of constraint violations per episode. Each bold line is average value over five random seeds, and each lighter color area means one fifth of standard deviation.

D. Ablation Study

To investigate the effect of the safeguard, we conduct ablation experiments in the point goal task with different numbers of the look-ahead cost networks. The number can be adjusted by the set $\mathcal{M}$ in (8), and $\{1\}$, $\{1, 6, 11\}$, $\{1, 6, ..., 21\}$, $\{1, 6, ..., 31\}$, and $\{1, 6, ..., 41\}$ are used for $\mathcal{M}$ in the experiments. Additionally, SGPO with no guide loss and $\mathcal{M} = \{1\}$ is evaluated. This version of SGPO is exactly same with the method proposed by Dalal *et al.* [15], denoted by *safety layer* (SL). The ablation results are presented in Figure 5f. As observed from the CV curves, the larger the number of the look-ahead, the lower the number of CVs. In addition, the scores according to the number of the look-ahead do not differ significantly, so it can be concluded that the number of the look-ahead does not cause performance degradation and can lower the number of CVs. Seeing the CV curve of the safety layer, the safety constraint is violated at the end of the training, which means the guide loss is critical to the prediction accuracy of the look-ahead cost networks.

VI. CONCLUSIONS

This paper proposes a novel safe RL method using the safeguard module which corrects an action safely in the exploration stage and makes the policy imitate the corrected action in the update stage. In simulation and experiments with a real robot, it is demonstrated that the proposed SGPO can satisfy the safety constraints with minimal interaction compared with other methods and achieves accomplished returns.

REFERENCES

[1] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine, "Solar: Deep structured representations for model-based reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7444–7453.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.

[3] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[4] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *J. Mach. Learn. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[5] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," in *Robotics: Science and Systems*, 07 2020.

[6] M. Wilson and T. Hermans, "Learning to manipulate object collections using grounded state representations," in *Proc. Conf. Robot Learn.* PMLR, 2020, pp. 490–502.

[7] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999, vol. 7.

[8] D. Ding, X. Wei, Z. Yang, Z. Wang, and M. Jovanovic, "Provably efficient safe exploration via primal-dual policy optimization," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 3304–3312.

[9] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," in *Proc. Int. Conf. Learn. Representations*, 2019.

[10] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6070–6120, 2017.

[11] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 22–31.

[12] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, 2018.

[13] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative safety critics for exploration," in *Proc. Int. Conf. Learn. Representations*, 2020.

[14] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, "Recovery rl: Safe reinforcement learning with learned recovery zones," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 4915–4922, 2021.

[15] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[16] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," *arXiv preprint arXiv:1910.01708*, 2019.

[17] C. R. Inc., "Jackal ugv - small weatherproof robot," https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/, September 2015.

[18] M. Ohnishi, L. Wang, G. Notomista, and M. Egerstedt, "Barrier-certified adaptive reinforcement learning with applications to brushbot navigation," *IEEE Trans. Robot.*, vol. 35, no. 5, pp. 1186–1205, 2019.

[19] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, 2019, pp. 3387–3395.

[20] L. Zhu, Y. Cui, and T. Matsubara, "Dynamic actor-advisor programming for scalable safe reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 10 681–10 687.

[21] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, vol. 31, 2018.

[22] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proc. Int. Conf. Autonomous Agents and MultiAgent Systems*, 2018, p. 2067–2069.

[23] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical programming*, vol. 27, no. 1, pp. 1–33, 1983.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.