2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Online Safety Property Collection and Refinement for Safe Deep Reinforcement Learning in Mapless Navigation

Luca Marzari[1,†,*], Enrico Marchesini[2,†] and Alessandro Farinelli[1]

***Abstract*— Safety is essential for deploying Deep Reinforcement Learning (DRL) algorithms in real-world scenarios. Recently, verification approaches have been proposed to allow quantifying the number of violations of a DRL policy over input-output relationships, called properties. However, such properties are hard-coded and require task-level knowledge, making their application intractable in challenging safety-critical tasks. To this end, we introduce the Collection and Refinement of Online Properties (CROP) framework to design properties at training time. CROP employs a cost signal to identify unsafe interactions and use them to shape safety properties. Hence, we propose a refinement strategy to combine properties that model similar unsafe interactions. Our evaluation compares the benefits of computing the number of violations using standard hard-coded properties and the ones generated with CROP. We evaluate our approach in several robotic mapless navigation tasks and demonstrate that the violation metric computed with CROP allows higher returns and lower violations over previous Safe DRL approaches.**

I. Introduction

Recently, Deep Reinforcement Learning (DRL) algorithms achieved significant results in robotic applications, ranging from manipulation [1], [2] to mapless navigation [3]–[5]. However, applying these techniques in real-world scenarios is seldom straightforward as non-linear function approximators are vulnerable to adversarial inputs [6], [7]. Given such issues, it is crucial to employ verification techniques and safety metrics in a safety-critical context.

To this end, Safe DRL techniques [8] have been investigated to enhance safety in robotic tasks. In particular, Safe DRL problems are typically modeled using Constrained Markov Decision Processes (CMPDs) [9], where an agent aims at maximizing a reward signal while keeping cost values accumulated upon visiting unsafe states under a hard-coded threshold. However, the constraints imposed by these approaches hinder exploration, failing to learn safe behaviors in complex environments [10], [11]. Alternative ways have been investigated to overcome the difficulty of designing Safe DRL algorithms that combine the concept of risk in the optimization while avoiding unsafe situations [8], [12], [13]. In particular, recent methods rely on Formal Verification (FV) [13] to quantify the number of correct decisions a policy chooses over desired safe specifications and use such information to evaluate the agents' decision-making. In more detail, a verification framework checks hard-coded input-output relationships (i.e., safety properties) in a domain of interest, verifying the decision-making process of a DRL policy. Hence, given a particular configuration of the agent's state space, i.e., a space $x \subset S$, a verification process identifies all the states where the policy selects the action $y \in A$ (where A is the agent's action space) that violates the safety properties. However, FV is an *NP-Complete* problem [14], and it is thus unfeasible to use such violation information to foster safety during the training.

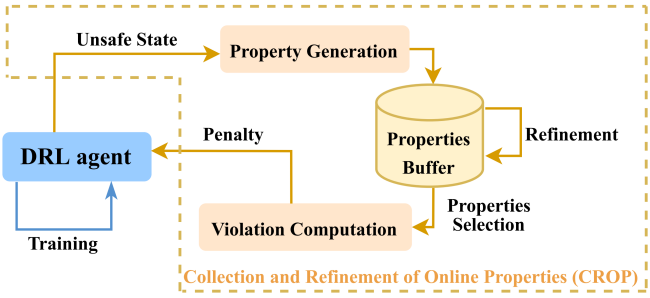


Fig. 1. Overview of the proposed CROP framework.

To this end, [11], [15] proposed a sample-based approximation method to enumerate the number of states in the space x that violate a specific property. Such a value referred to as *violation*, has been used to induce safety information during the training. The *violation* is particularly beneficial in a safety-critical setup, as it includes safety information based on the hard-coded specified properties, thus, allowing foster desired safe behaviors in the agent.

Modeling hard-coded properties, however, is a severe limitation of verification approaches for two main reasons. First, the properties assume having precise task-level knowledge regarding the robotic task (e.g., presence of fixed or dynamic obstacles, their shape, etc.) and agent specification (e.g., in the case of a mobile robot, the agent's dimension, linear and angular velocity $\vec{v}$). Second, writing an exhaustive set of properties to cover all the unsafe behaviors is impractical in dynamic scenarios. These limitations significantly impact applying the *violation* in complex safety-critical domains where writing safety properties is not trivial. For example, it could be unfeasible to write properties for robots with complex dynamics (e.g., quadrupeds with multiple joints), as shown in [11]. For this reason, prior works that consider the violation in DRL contexts [16]–[18] use a limited set of 3-5 high-level properties to provide the agent with core safe behaviors specification (e.g., in the case of robotic navigation, do not turn right if close to an obstacle). Hence, the agent may experience unsafe interactions not encoded by the manually specified set of safety properties.

[1] Department of Computer Science, University of Verona, Italy.
[2] Khoury College of Computer Sciences, Northeastern University, US.
[†] Equal contribution
[*] Contact author: luca.marzari@univr.it

Despite the limitations, previous violation-based approaches [19] achieved comparable or better results over commonly employed Safe DRL approaches such as Constrained DRL [20]. Motivated by this, we address the issue of hard-coding properties by providing a method that can collect and refine safety specifications during training.

Specifically, we propose the Collection and Refinement of Online Properties (CROP) framework (Fig. 1) to collect and refine safety properties during training time. Similarly to prior Safe DRL literature [21]–[23], CROP uses an indicator function, called cost, that deems a state-action interaction as unsafe. When the agent performs an unsafe interaction, CROP generates a safety property using the state and the action that led to the hazardous situation. Hence, CROP performs a *refinement* procedure to combine similar properties collected online. In particular, CROP uses a similarity rule to compute whether two safety properties encode similar unsafe interactions. Hence, our method merges similar properties into one that incorporates both the unsafe states. Finally, the violation is computed using a previous sample-based approximation method [11] on CROP's dynamic set of online generated properties and is used as a penalty for the training.

Following recent works on Safe DRL [10], [11], [24], [25], we focus our evaluation on different mapless navigation domains and compare our method with (i) a penalty-based method that uses hard-coded safety properties, and (ii) a Constrained DRL algorithm. Our evaluation relies on realistic Unity environments [26] that enable the transfer of policies trained in simulation on ROS-enabled platforms. We provide a video (`shorturl.at/drVW6`) with the real-world experiments of the policies trained in our environments using the TurtleBot3 as an agent. The empirical evaluation confirms that the CROP framework addresses the lack of information provided by the hard-coded properties approaches. Crucially, our results show that using online generated properties lead to a more robust *violation* computation that translates into safer behaviors (i.e., reduced number of collision during the training and evaluation in a previously unseen scenario). Moreover, we provide formal guarantees on the policy behaviors by employing a formal verification analysis of the trained agents [27].

This work makes the following contributions to the state-of-the-art: (i) we propose CROP, a novel framework for collecting properties online, overcoming the limitations of hard-coding such properties. (ii) We show how to refine similar safety properties to limit the number of generated properties and improve the design of the input-output relationships. (iii) We provide a quantitative empirical evaluation with prior approaches based on hand-designed properties and a standard Constraint DRL method. Moreover, we perform a qualitative validation on the real robot.

II. Preliminaries and Related Work

A. Safety property

Safety properties are input-output relationships. In more detail, given a Deep Neural Network (DNN) $\mathcal{F}$ with $y_{1,\dots,n}$ possible outputs and a subspace of the state space $x \subset S$, a property is defined as [13]:

$$\mathcal{P} : \underbrace{x \subset S}_{\mathcal{P}[x]} \implies \underbrace{y_k < y_i}_{\mathcal{P}[y]} \quad \forall i \in [1, n] - \{k\} \tag{1}$$

Typically, the subspace x is encoded as a property $\mathcal{P}$ (i.e., $\mathcal{P}[x]$) using intervals to shape the surroundings of an unsafe state of interest. Hence, $\mathcal{P}[x]$ is a vector of intervals, one for each value of the observation space S. Given our interest in verifying DRL policies, the inequality of Eq. 1 $\mathcal{P}[y]$, encodes a condition as *never select the action corresponding to the output* y_k. For instance, in a value-based framework where the policy chooses the action with the highest value, the post-condition is implemented to verify that the selected action does not correspond to the action that would violate the safety property. A similar check can be achieved in a policy gradient setup by using deterministic policies for evaluation and incorporating stochastic policies for exploration. For example, prior works [17], [28] in mapless navigation tasks use the agent's maximum velocity $\vec{v}$ to compute the minimum distance that can be reached before colliding with an obstacle. Fig.2 (right) shows a practical example where, knowing that a distance greater than $\boldsymbol{\epsilon} = 0.05$ prevents the agent from colliding at maximum speed, a safety property checks that an agent does not move forward while there is an obstacle in front. Following Eq. 1, this is formally encoded as:

$$\mathcal{P}_{\uparrow} : x_0, x_4 \in [0.95, 1] \land x_1, x_3 \in [0.03, 0.08] \land \underbrace{x_2 \in [0, 0.05] \land x_5, x_6 \in [-1, 1]}_{\mathcal{P}_{\uparrow}[x]} \implies \underbrace{y_0 < y_1, y_2}_{\mathcal{P}_{\uparrow}[y]}$$

where $x_0, \dots, x_4$ are the 5 lidar values, x_5, x_6 is the relative position of the goal with respect to the agent's position (i.e., distance and heading), and action y_0 corresponds to a forward movement. In the example, we set x_5, x_6 equal to $[-1, 1]$ to indicate a generic goal position in the environment. Broadly speaking, if the agent is in a state described by $\mathcal{P}_{\uparrow}[x]$, it should not choose action y_0 that corresponds to a forward movement (hence, leading to a collision).

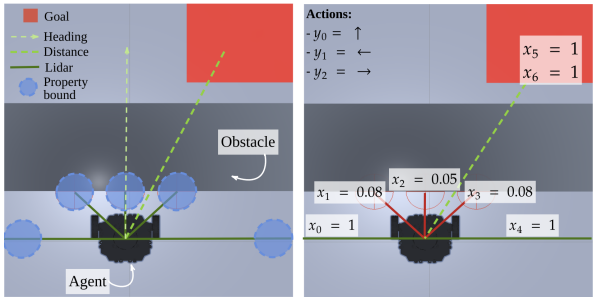


Fig. 2. Left: components of a safety property. Right: Explanatory image of a safety property $\mathcal{P}_{\uparrow}$ for a mapless navigation context.

B. Sample-based violation computation

Due to the computational demands of the verification frameworks [13], and the *NP-Completeness* of DNN Formal Verification (FV) [14], computing the *violation* value in

the training loop is unfeasible. To this end, we compute an approximate *violation* using a sample-based method to enumerate the number of states in the subset x that violate a property $\mathcal{P}$, as proposed by [15], [29]. In detail, these approaches require $\mathcal{F}$ and a set $\mathbb{B}$ of safety properties that contain, in the subspace x, a surrounding of an unsafe state. For each property's subspace $\mathcal{P}[x] \in \mathbb{B}$, we sample a set of m random points (i.e., states). These points represent a set of possible unsafe states the agent could reach in that subspace $\mathcal{P}[x]$. Finally, we perform a DNN forward propagation of these unsafe states and quantify the ratio of unsafe decisions of a policy π over the number of sampled points.

C. Penalty-based objective for safety

We model the navigation tasks as a Markov Decision Process (MDP), described by a tuple $< S, A, R, P, \gamma >$ where S is the observation space, A is the action space, $P : S \times A \to S$ is the transition function and $R : S \times A \to \mathbb{R}$ is the reward function. $\gamma \in [0, 1]$ is the discount factor that allows the control of the influence of future rewards. Given a stationary policy $\pi \in \prod$, the agent aims to maximize $\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, i.e., the expected discounted return for each trajectory $\tau = (s_0, a_0, \cdots)$.

Recent works [8], [30], [31] show how incorporating penalties into the reward effectively influences policy toward safer behaviors while avoiding the limitations introduced by constrained approaches [22]. Following this direction, we incorporate the *violation* value in a penalty-based objective:

$$\max_{\pi\in\Pi} J_R^{\pi} := \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) - Z(\cdot)\right] \qquad (2)$$

where $Z(\cdot)$ is a penalty function (the cost or the *violation*).

III. Collection and Refinement of Online Properties

The general flow of CROP is presented in Algorithm 1. In particular, we augment the DRL training with a buffer $\mathbb{P}$ that stores all the properties generated by CROP (line 3). In more detail, given a state s_t deemed unsafe by the cost signal, CROP generates a new safety property $\mathcal{P}'$ considering an $\boldsymbol{\epsilon}$ surrounding of the state s_{t-1}, and the action a_{t-1} that triggered a positive cost (i.e., considering the state-action interaction as unsafe) (line 5). In particular, CROP models the subspace of $\mathcal{P}'[x]$ with a set of intervals written as $[x_i - \boldsymbol{\epsilon},\ x_i] \quad \forall\ x_i \in s_{t-1}$. The implication $\mathcal{P}'[y]$ checks that the agent does not choose the action a_{t-1} that led to the unsafe state s_t. Note that the $\boldsymbol{\epsilon}$ required by CROP differs from the epsilon value required to design a hard-coded safety property (as in the example of Sec.II-A), which requires task-level knowledge. Specifically, in CROP, $\boldsymbol{\epsilon}$ is initialized as an arbitrarily small value that will be shaped during refinement. Subsequently, CROP selects the set of properties that contain the state s_{t-1} in their subspaces $\mathcal{P}[x]$ and that encode the implication $\mathcal{P}[y]$ over the same action a_{t-1} (line 6). Hence, for each property selected, our method verifies whether there is a similar property $\mathcal{P}$ with respect to $\mathcal{P}'$ that needs to be refined (lines 7-14). In particular, CROP uses the refinement

Algorithm 1 Collection and Refinement of Online Properties

1: **Given:**
- a DRL agent parametrized by a DNN $\mathcal{F}_\theta$
- $\boldsymbol{\epsilon}$ an initial size for a surrounding of an unsafe state to consider
- $\sim$ rule of similarity for safety properties as in Sec.III-A.

2: During each episode of the training of DRL agent:
3: $\mathbb{P} \leftarrow \emptyset$
4: **if** s_t is unsafe (i.e., cost > 0) **then**
5: $\quad\mathcal{P}' \leftarrow$ *Generate Property*($\boldsymbol{\epsilon}$, s_{t-1}, a_{t-1}) ▷ as Sec.III
6: $\quad$**for** $\mathcal{P} \in \mathbb{P}$ where $s_{t-1} \subseteq \mathcal{P}[x] \wedge a_{t-1} = y_k \in \mathcal{P}[y]$ **do**
7: $\quad\quad$**if** $\nexists\ \mathcal{P} \sim \mathcal{P}'$ **then**
8: $\quad\quad\quad\mathbb{P} \leftarrow \mathbb{P} \cup \mathcal{P}'$
9: $\quad\quad$**else**
10: $\quad\quad\quad\mathbb{P} \leftarrow$ *Property Refinement*($\mathcal{P}, \mathcal{P}', \mathbb{P}$) ▷ as Sec.III-A
11: $\quad\quad$**end if**
12: $\quad$**end for**
13: $\quad$violation $\leftarrow$ *violation computation*($\mathcal{F}_\theta$, $\mathbb{P}$) ▷ as in Sec.II-B
14: **end if**

rule described in the next section to consider the similarity between properties. However, the collection of properties following this procedure remains challenging in domains where many unsafe interactions occur (i.e., the buffer $\mathbb{P}$ grows significantly). Hence, we decide to reset the properties buffer at the beginning of each episode. This approach allows us to compute the *violation* to be given as a penalty only on the unsafe situations encountered during the trajectories seen during the episode. Finally, we compute the *violation* using the sample-based estimation discussed in Section II-B (line 16).

A. Properties similarity and refinement process

Our method requires a rule to define when two safety properties are *not similar*. To this end, we illustrate a possible scenario in the context of mapless navigation in Fig. 3. In particular, given the nature of a safety property, it is conceivable to find two or more different safety properties for the same action y_k of a DNN. As an example, in both the scenarios depicted in Fig. 3, the agent would collide upon moving forward. However, the subspaces of the safety properties $\mathcal{P}[x]$ significantly differ (for simplicity, we omit the values for heading and distance from goal). Hence, we detect *not similar* properties leveraging Moore's interval algebra [32]. In particular, we define this rule:

$$\begin{aligned}\mathcal{P} \not\sim \mathcal{P}' \iff \exists\ i \in [0, |x|]\ s.t \\ \big(\mathcal{P}[x]_i \subseteq \psi \vee \mathcal{P}'[x]_i \subseteq \psi\big) \wedge \\ |\mathcal{P}[x]_i - \mathcal{P}'[x]_i| > \beta\end{aligned}$$

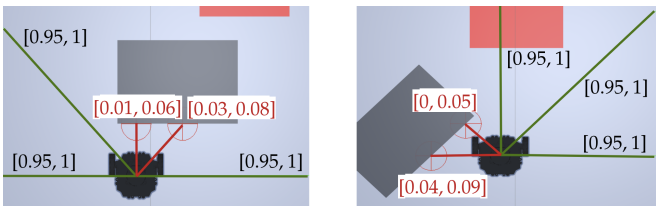
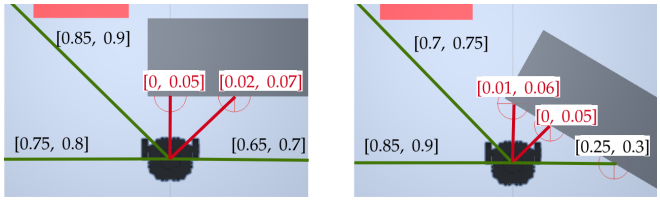


Fig. 3. Explanatory example of two different safety properties for the same action y_k that encodes a forward movement.

Broadly speaking, given two safety properties $\mathcal{P}$ and $\mathcal{P}'$, the above rule investigates whether there is at least one index $i \in [0, |x|]$ of the properties subspaces $\mathcal{P}[x]_i$ or $\mathcal{P}'[x]_i$, which is contained in ψ (i.e., an interval that encodes a potentially unsafe situation. For example, in Fig. 3 on the right, ψ equals [0, 0.09] and is highlighted in red). Moreover, CROP checks whether the lower bound of the absolute difference in value between the two considered intervals $|\mathcal{P}[x]_i - \mathcal{P}'[x]_i|$ is greater than a threshold value β. If these two conditions are true for at least one index i in the properties, then we consider $\mathcal{P}$ and $\mathcal{P}'$ *not similar*. The value β can be tuned to be more or less restrictive about considering two properties *not similar*. Our similarity rule successfully distinguishes two different properties in the situations depicted in Fig. 3. For example, for the first lidar scan on the left, we have at least one of the two intervals in $[0, 0.09]$ and the lower bound of the absolute difference between the two intervals is equal to 0.86 and, therefore, above a threshold $\beta = 0.1$. Crucially, similar considerations are applicable to different tasks to apply our collection and refinement procedure.

On the other hand, if two properties are *similar*, the refinement process is performed, merging the two properties into one, considering the minimum lower and maximum upper bound for each interval. An explanatory image is reported in Fig. 4.



$$\mathcal{P}_{\uparrow} : x_0 \in [0.75, 0.9] \wedge x_1 \in [0.7, 0.9] \wedge x_2 \in [0, 0.06] \wedge x_3 \in [0, 0.07] \wedge x_4 \in [0.25, 0.7] \implies y_0 < y_1, y_2$$

Fig. 4. Explanatory example of refinement process for two *similar* safety properties.

IV. Experiments

Our evaluation focuses on a mapless navigation task, a well-known benchmark in the recent Safe DRL literature [10], [11], [24], [25]. In particular, in this task, a robot has to learn to navigate towards a random target using only local observation and the target position, without a map of the surrounding environment or obstacles. Similar to prior works [17], [25], we consider an observation space with 21 sparse laser scans sampled at a fixed distance in a 360 degrees range and two values for the target position relative to the robot (i.e., distance and heading) as observation space. In addition, to reduce training times while maintaining good navigation skills, we use a discrete action space [25] that encodes angular and linear velocities. In our experiment, we use six discrete actions to encode the following movements: a forward movement, a rotation with and without linear velocity, and a stationary position. This allows us to capture the different types of movements accurately.

We implement CROP in a policy-based Proximal Policy Optimization (PPO) [33] baseline using a penalty shaping as in Eq. 2. We compare our method with previous violation-based penalty approaches that use hand-designed properties (i.e., PPO_violation), a cost-based penalty (i.e., PPO_cost), and a standard constraint approach, i.e., Lagrangian PPO (LPPO) [22]. We use standard hyper-parameters from earlier research [10], [11], [33] consisting of two hidden layers with 64 units each and activated with ReLU to model actors and critic networks, respectively. Moreover, to ensure a fair comparison with a standard constraint approach (LPPO), we set the cost threshold to the average cost produced by the penalty-based baselines. All the evaluated approaches use the same base reward function consisting of a dense reward determined by the difference in distance (Δ) between the target and the agent's position. Regarding previous PPO_violation approach, we consider a set of hard-coded proprieties to test basic safety behaviors, such as *do not perform a {forward, left, right} movement, if there is an obstacle near to the {back, front, left right}*. Specifically, we consider the same properties of prior work [27] plus the properties to cover the back of the robot due to dynamic obstacles. Moreover, for PPO_CROP we set β equal to 0.1 as it achieves a good trade-off between the number of properties collected and the safety information provided to the agent. We also tuned a cloud size of $m = 10.000$ points and we reset the buffer of properties at each episode as in Alg.1 for our CROP method. Data are collected on a RTX 2070, and an i7-9700k, reporting the mean and standard deviation gathered from ten independent runs.

Our empirical evaluation aims at investigating whether the CROP-based approach can increase return and reduce the number of violations. Crucially, a lower number of violations translate into fewer collisions (i.e., the agent chooses fewer actions that potentially lead to a collision). Hence, a lower violation value directly maps into a lower cost.

A. Environments

We use two training environments to evaluate our methodologies, called *Fixed obstacles* and *Dynamic obstacles*, respectively.

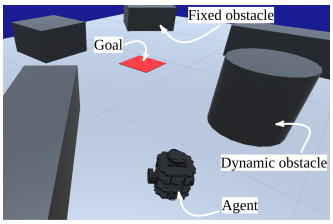
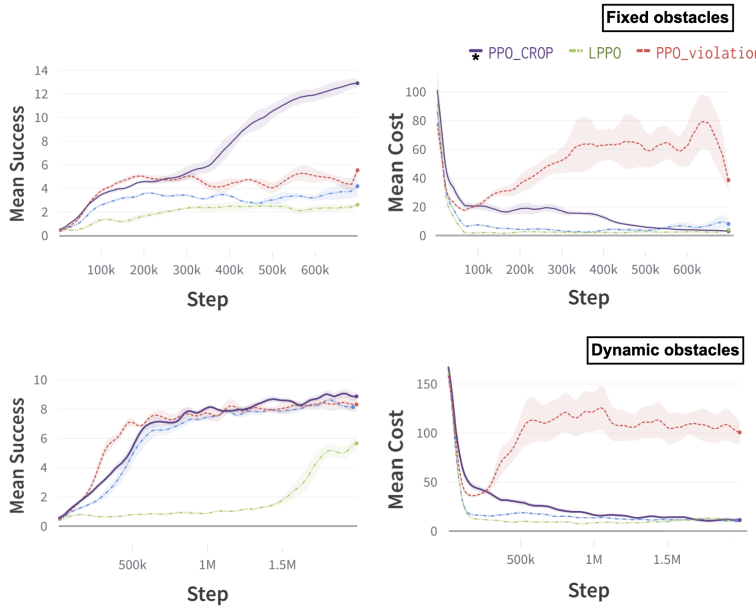


Fig. 5. Overview of the *Evaluation* environment

More specifically, the environments differ for obstacles that can be *Fixed* (parallelepiped-shaped static objects) or *Dynamic* (cylindrical-shaped objects that move to random positions with constant velocity). The environments have *non-terminal* obstacles, i.e., they return a signal upon each collision, but the episode does not terminate (i.e., the robot has to learn how to avoid getting stuck into the obstacles). The algorithms use such collisions to trigger a positive



Mean Violation	100k	300k	700k
*PPO_CROP	6.3 ± 1.5	6.4 ± 1.8	**0.83 ± 0.4**
LPPO	**1.1 ± 0.5**	**2.1± 1.4**	3.5 ± 1.2
PPO_violation	4.8 ± 1.1	2.7 ± 0.6	0.92 ± 0.1
PPO_cost	3.7 ± 1.0	2.5 ± 0.4	6.7 ± 5.8

Mean Violation	500k	1M	2M
*PPO_CROP	16.2 ± 2.8	10.9 ± 1.7	**7.8 ± 3.7**
LPPO	**8.9 ± 1.4**	**7.1± 1.4**	9.7 ± 2.2
PPO_violation	10.2 ± 3.5	10.4 ± 4.3	13.4 ± 6.0
PPO_cost	15.8 ± 2.9	11.5 ± 1.1	9.8 ± 1.2

Fig. 6. Average success, cost, and violation for our versions of PPO_cost, PPO_violation, *PPO_CROP and LPPO. * indicates our method.

value or other desired penalties. Dynamic obstacles are particularly challenging because they lead to a non-stationary environment and make the problem partially observable [34]. To this end, we extend the setup of previous work [11], [25] to consider 360-degree scans. This setup allows us to obtain a good trade-off between mean success and obstacle avoidance behaviors. Finally, we test the generalization skills of the trained policies in an *Evaluation* environment that is not employed during training, depicted in Fig. 5.

B. Motivating example

We conducted an experiment to confirm our hypotheses regarding the limitation of hard-coded properties-based approaches in the fixed obstacle domain. In particular, we compare the number of properties employed to compute the violation value (i.e., the properties that contains the unsafe interaction in their domain-codomain) in the case of PPO_CROP (ours) and PPO_violation (which considers a set of five hard-coded properties for navigation as in previous work [27]). In this experiment, we expect PPO_CROP to collect a higher number of properties that describe the actual unsafe behaviors experienced during the training. Hence, the number of properties used to compute the violation in the case of PPO_CROP should be significantly higher. As discussed in the following empirical evaluation, such a higher number of properties lead to a more informative *violation* value (i.e., it is computed considering a higher number of unsafe situations), hence improving the safety aspect of the trained policies.

Fig.7 shows the number of properties used by PPO_violation and PPO_CROP to compute the *violation*. As expected, CROP collects a high number of properties on the task, while prior work cannot cope with the complexity of the environment and employs a reduced number of properties for computing the *violation*.

Finally, we note that the number of properties used in CROP tends to increase by the end of the training, while it remains constant for the other methodology. Such result implies that collecting properties during the training by the agent allows it to find borderline cases of unsafe situations that are difficult to explore using hand-designed properties.

C. Empirical training evaluation

Fig.6 shows the results of our evaluation in *Fixed obstacles* and *Dynamic obstacles*. In more detail, in the *Fixed obstacles* environment, our PPO_CROP outperforms other methodologies in terms of the average success rate and significantly reduces the cost (i.e., number of collisions) and *violation* during the training. In this environment, PPO_CROP at convergence shows a $\approx 0.09\%$ and a $\approx 5.87\%$ *violation* improvement over the PPO_violation and PPO_cost counterparts. Considering the total number of points used to compute the violation (which depends on the number of properties employed by the approaches), such improvements map to 180 and 11740 fewer collisions, respectively. Regarding the *Dynamic obstacles* environment, our methodology achieves better or comparable successes and cost values

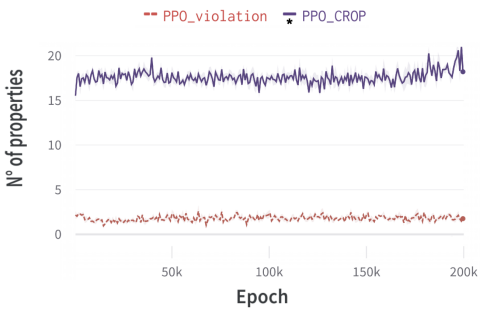


Fig. 7. Average number of proprieties' used for the *violation* computation during the training of PPO_violation and our PPO_CROP.

TABLE I

MEAN VIOLATION EXPRESSED IN PERCENTAGE VALUES COMPUTED USING FORMAL VERIFICATION FOR EACH MODELS AT CONVERGENCE FOR *Dynamic obstacles* (LEFT) AND *Fixed obstacles* (RIGHT) ENVIRONMENTS.

	Method			
Property	**PPO_cost**	***PPO_CROP**	**PPO_violation**	**LPPO**
$\mathcal{P}_{\uparrow}$	0.2 ± 0.04	0.09 ± 0.07	0.15 ± 0.1	0.29 ± 0.1
$\mathcal{P}_{\rightarrow}$	0.47 ± 0.04	0.45 ± 0.1	0.41 ± 0.2	0.38 ± 0.1
$\mathcal{P}_{\leftarrow}$	0.51 ± 0.2	0.43 ± 0.08	0.39 ± 0.1	0.43 ± 0.3
$\mathcal{P}_{\nearrow}$	0.49 ± 0.09	0.36 ± 0.08	0.46 ± 0.1	0.49 ± 0.3
$\mathcal{P}_{\nwarrow}$	0.5 ± 0.1	0.33 ± 0.07	0.42 ± 0.2	0.44 ± 0.2
SUM	2.17	**1.66**	1.83	2.03

	Method			
Property	**PPO_cost**	***PPO_CROP**	**PPO_violation**	**LPPO**
$\mathcal{P}_{\uparrow}$	0.3 ± 0.1	0.26 ± 0.2	0.4 ± 0.2	0.28 ± 0.1
$\mathcal{P}_{\rightarrow}$	0.52 ± 0.1	0.38 ± 0.1	0.35 ± 0.2	0.34 ± 0.3
$\mathcal{P}_{\leftarrow}$	0.63 ± 0.3	0.42 ± 0.2	0.35 ± 0.1	0.63 ± 0.4
$\mathcal{P}_{\nearrow}$	0.57 ± 0.2	0.35 ± 0.1	0.39 ± 0.2	0.57 ± 0.3
$\mathcal{P}_{\nwarrow}$	0.56 ± 0.2	0.33 ± 0.1	0.4 ± 0.2	0.4 ± 0.3
SUM	2.58	**1.74**	1.89	2.22

over the counterparts but significantly reduces the violations during the training, showing a $\approx 5.60\%$ and a $\approx 2.00\%$ *violation* improvement, which corresponds on average to 11200 and 4000 fewer collisions. For a fair evaluation, all the algorithms are trained over the same parameters and configurations of dynamic obstacles (i.e., they experience the same random obstacle movements during the training phases, which changes over the different seeds). Finally, LPPO satisfies the constraint imposed on the cost threshold, set respectively at 5 and 12 in the *Fixed obstacles* and *Dynamic obstacles* environments. However, it achieves the lowest number of successes, confirming the performance trade-off in complex scenarios and the difficulty of tuning the value of the multiplier parameter underlined in other recent research works [10], [11]. Crucially, the lower number of successes is not motivated by performing longer but safer paths, as PPO_CROP also achieves better results in terms of safety (i.e., lower violations and cost).

D. Evaluation

In order to test the generalization skills of the trained policies in a previously unseen scenario, we perform an additional experiment on the *Evaluation* environment (reported in Fig.5). Tab. II reports each model's average success, cost, and *violation* at convergence. Results confirm the benefit of using our CROP, as PPO_CROP shows superior navigation skills by achieving a higher number of successes while being safer over LPPO and the cost counterpart.

TABLE II

EVALUATION RESULTS IN THE *Evaluation* ENVIRONMENT

Method	**Mean Success**	**Mean Cost**	**Mean Violation**
PPO_cost	8.4 ± 1.5	2.4 ± 0.5	2.1 ± 0.5
*PPO_CROP	**8.8** ± 0.8	**1.8** ± 0.9	**1.3** ± 0.6
PPO_violation	8.2 ± 0.6	84.9 ± 28.6	79.7 ± 30.7
LPPO	4.0 ± 0.9	2.1 ± 1.6	1.8 ± 1.3

E. Formal Verification at convergence

We provide formal guarantees on the trained policies behaviors using recent FV literature [27] in reinforcement learning scenarios. In contrast to the sample-based estimation used during the training, formal verification framework provably ensures the policy decisions in the situations specified in safety properties (i.e., it does not depend on the samples, but it is an exhaustive search over the entire space described by the properties' domains), therefore it provides guarantees that the behavior of the trained model respects a set of safety properties. Moreover, such formal verification frameworks also return all the states where a policy violates the properties. For a fair evaluation of the policies at convergence, we verify all the policies using the five hard-coded properties of prior work [11]. Tab. I shows the average violations computed over such properties. Crucially, the formal analysis of the trained networks confirms our empirical results. Overall, PPO_CROP results the safest approach, with the lowest *violation* rate in both the *Dynamic* and *Fixed* obstacles environment, which translates into fewer collisions.

F. Real-Robot experiments

The Unity framework [26] allowed us to transfer the policies trained in simulation on ROS-enabled platforms such as our Turtlebot3 (Fig. 8). In particular, we show a comparison between our PPO_CROP and PPO_violation in several real corner case scenarios[1].

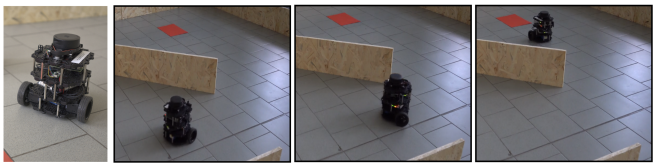


Fig. 8. Real-world experiments.

V. DISCUSSION

This paper addresses the limitations of safety-oriented approaches that leverage only hard-coded properties. The proposed framework, CROP, allows the collection and refinement of safety properties during the training, thus overcoming the limitations of hand-designed approaches. We show that CROP allows us to obtain a more robust approach with respect to other Safe DRL methodologies, promoting safer behaviors while maintaining similar or better returns.

Future directions involve extending the use of CROP where safety is of pivotal importance, such as multi-agent systems where the properties have to consider cooperative situations.

[1]Video `shorturl.at/drVW6` of real-world experiments.

REFERENCES

[1] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[2] L. Marzari, A. Pore, D. Dall'Alba, G. Aragon-Camarasa, A. Farinelli, and P. Fiorini, "Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks," in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 640–645.

[3] E. Marchesini and A. Farinelli, "Enhancing deep reinforcement learning approaches for multi-robot navigation via single-robot evolutionary policy search," in *International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5525–5531.

[4] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real drl: Continuous control of mobile robots for mapless navigation," in *IROS*, 2017.

[5] E. Marchesini and A. Farinelli, "Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 4583–4588.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[7] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz, "Verifying learning-based robotic navigation systems," *arXiv preprint arXiv:2205.13536*, 2022.

[8] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," in *Journal of Machine Learning Research (JMLR)*, 2015.

[9] E. Altman, "Constrained markov decision processes," in *CRC Press*, 1999.

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

[11] E. Marchesini, D. Corsi, and A. Farinelli, "Exploring safer behaviors for deep reinforcement learning," in *AAAI Conference on Artificial Intelligence Conference on Artificial Intelligence*, 2022.

[12] E. Marchesini and C. Amato, "Safety-informed mutations for evolutionary deep reinforcement learning," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, p. 1966–1970.

[13] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Foundations and Trends® in Optimization*, 2021.

[14] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International conference on computer aided verification*. Springer, 2017, pp. 97–117.

[15] E. Marchesini, L. Marzari, A. Farinelli, and C. Amato, "Safe deep reinforcement learning by verifying task-level properties," in *nternational Conference on Autonomous Agents and Multiagent Systems (AAMAS),*, 2023.

[16] E. Marchesini, D. Corsi, and A. Farinelli, "Benchmarking safe deep reinforcement learning in aquatic navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 5590–5595.

[17] L. Marzari, D. Corsi, E. Marchesini, and A. Farinelli, "Curriculum learning for safe mapless navigation," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 766–769.

[18] A. Pore, D. Corsi, E. Marchesini, D. Dall'Alba, A. Casals, A. Farinelli, and P. Fiorini, "Safe reinforcement learning using formal verification for tissue retraction in autonomous robotic-assisted surgery," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4025–4031.

[19] W. Zhao, T. He, and C. Liu, "Model-free safe control for zero-violation reinforcement learning," in *Proceedings of the 5th Conference on Robot Learning*, 2022, pp. 784–793.

[20] Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021, pp. 4508–4515.

[21] Y. Liu, J. Ding, and X. Liu, "IPO: interior-point policy optimization under constraints," in *AAAI Conference on Artificial Intelligence*, 2020.

[22] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by pid lagrangian methods," in *International Conference on Machine Learning (ICML)*, 2020.

[23] J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. Pal, "Direct behavior specification via constrained reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2022.

[24] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IROS*, 2017.

[25] E. Marchesini and A. Farinelli, "Discrete deep reinforcement learning for mapless navigation," in *ICRA*, 2020.

[26] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A platform for intelligent agents," in *CoRR*, 2018.

[27] D. Corsi, E. Marchesini, and A. Farinelli, "Formal verification of neural networks for safety-critical tasks in deep reinforcement learning," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.

[28] E. Marchesini, D. Corsi, and A. Farinelli, "Genetic soft updates for policy evolution in deep reinforcement learning," in *ICLR*, 2021.

[29] L. Marzari, D. Corsi, F. Cicalese, and A. Farinelli, "The #dnn-verification problem: Counting unsafe inputs for deep neural networks," *ArXiv*, vol. abs/2301.07068, 2023.

[30] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99, 1999, pp. 278–287.

[31] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[32] R. E. Moore, "Interval arithmetic and automatic error analysis in digital computing," Stanford Univ Calif Applied Mathematics And Statistics Labs, Tech. Rep., 1962.

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

[34] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*, 1st ed. Springer Publishing Company, Incorporated, 2016.