Safe and Efficient Operation with Constrained Hierarchical Reinforcement Learning

Felippe Schmoeller Roza
Fraunhofer IKS
Munich, Germany

Karsten Roscher
Fraunhofer IKS
Munich, Germany

Stephan Günnemann
Technical University of Munich
Munich, Germany

Abstract

Hierarchical Reinforcement Learning (HRL) holds the promise of enhancing sample efficiency and generalization capabilities of Reinforcement Learning (RL) agents by leveraging task decomposition and temporal abstraction, which aligns with human reasoning. However, the adoption of HRL (and RL in general) to solve problems in the real world has been limited due to, among other reasons, the lack of effective techniques that make the agents adhere to safety requirements encoded as constraints, a common practice to define the functional safety of safety-critical systems. While some constrained Reinforcement Learning methods exist in the literature, we show that regular flat policies can face performance degradation when dealing with safety constraints. To overcome this limitation, we propose a constrained HRL topology that separates planning and control, with constraint optimization achieved at the lower-level abstraction. Simulation experiments show that our approach is able to keep its performance while adhering to safety constraints, even in scenarios where the flat policy's performance deteriorates when trying to prioritize safety.

1 Introduction

Reinforcement Learning (RL), despite not being a new paradigm, has recently reemerged as a viable alternative to tackle problems that are especially difficult solve using other Machine Learning (ML) approaches. By gathering experience and learning while interacting with the environment, RL was able to achieve robot control [31], human-level video-game playing [18, 28] and even defeated the world champion at the game of go [25]. Amongst a plethora of approaches within the RL field, Hierarchical Reinforcement Learning (HRL) has gained attention as a potential solution to improve on known limitations of RL agents, including sample efficiency, scalability, generalization, and being better at solving problems with long-term credit assignment and sparse rewards [26, 7]. HRL is inspired by biological brains and leverages the concept of temporal abstraction, where complex tasks are decomposed into a hierarchy of subtasks, allowing the agent to reason at different levels of granularity, facilitating efficient exploration, and enabling transfer learning.

Despite such achievements and potential, a gap still exists regarding how to transfer HRL (and RL in general) models to real-world systems, especially when considering safety-critical applications, hindering the possibility of deploying such intelligent systems to automate on tasks that still rely on human decision and control. Among different reasons, RL models generally lack strong robustness and safety guarantees, especially when Deep Neural Networks (DNNs) are used as function approximators [17]. RL agents, when left to explore and learn without explicitly accounting for safety aspects, may encounter situations where a wrong action can lead to catastrophic consequences. In domains such as healthcare, finance, and transportation, ensuring safety is essential, with bad decisions or actions potentially incurring severe implications for the surrounding environment and even for human lives.

16th European Workshop on Reinforcement Learning (EWRL 2023).

Safe RL methods, that seek to mitigate various concerns and challenges related to the safety of intelligent systems, can be classified in different categories. For instance, safe exploration focuses on the design of systems that ensure safety during the training phase of RL systems [19]. Constrained RL concentrates on learning policies that respect a given set of constraints [2]. Shielding involves the addition of a safety wrapper that verifies during runtime if the actions taken by the agent comply with a set of safety requirements [4]. However, in most cases, these approaches are designed around traditional flat-policy models. With the growing interest in leveraging hierarchical models to solve problems that involve complexity and multi-level reasoning, it becomes crucial to find ways to incorporate safety measurements into HRL architectures too. This integration opens up new possibilities for leveraging the benefits of hierarchical reasoning and brings RL closer to being a viable and effective solution for addressing real-world challenges.

The primary objective of this paper is to address the challenge of integrating constraint satisfaction capabilities in HRL agents to keep a safety signal bounded. That is done by extending goal-conditioned algorithms by adding a safety layer that uses constraint dynamics models to predict safety violations and to project the actions into a safe set. By disentangling planning and control and enforcing safety constraints at the lower-level policy, our approach aims to achieve a better balance between performance and safety when compared against a flat-policy baseline. We also show how the baseline fails at learning in some scenarios when trying to keep the safety requirements. The experimental results demonstrate the efficacy of our approach in achieving both safety and competitive performance. The findings of this study have important implications for real-world applications, where safe and reliable RL agents are essential.

The rest of the paper is organized as follows. section 2 provides an overview of the related work in safe RL and hierarchical RL with the preliminaries and background presented in section 3. section 4 explains the methodology and formulation of our HRL framework. section 5 presents the experimental setup and results, followed by a discussion in section 6. Finally, section 7 concludes the paper and highlights potential directions for future research.

2 Related Work

Safe RL. Safety is a crucial aspect in RL, particularly in scenarios where wrong decisions can have severe consequences. The main aspects of Safe RL related to this paper are safe exploration and constrained RL. Safe exploration involves designing RL agents able to explore the environment within a predefined while still allowing for trial-and-error exploration, necessary to learn optimal or sub-optimal policies. [12] pinpoints several attempts to derive safe techniques that oppose traditional unsafe explorations/exploitation methods, including using teacher advice [14], providing initial knowledge [14], and through risk-directed exploration [13].

On the other hand, constrained RL addresses the dual problem of maximizing utility while keeping a safety signal under acceptable boundaries [5]. These different objectives might be conflicting, which is often the case when dealing with safety constraints. Different approaches have been proposed to achieve constrained control with RL. In [8] (and similarly in [22]), a safety wrapper is proposed to allow safe exploration in continuous action spaces. The method consists of learning the constraint dynamic models offline and using these models to predict future constraint values. If the proposed action does not comply with the constraint predictions of the learned models an alternative safe action is calculated. In [29], the authors propose SNO-MDP, a framework that uses Gaussian processes to characterize the unknown constraint functions and explore the safety region before optimizing over the cumulative reward. Constrained Policy Optimization (CPO) is another popular algorithm for training RL in constrained environments, presenting guarantees for near-constraint satisfaction [2].

Hierarchical Reinforcement Learning. Different takes on HRL are present in the literature. The Feudal RL architecture is composed of a manager, that chooses a direction to go in a latent state space, and workers, that learn to follow that direction through actions in the environment [9, 27]. Another popular HRL approach is the options framework, which formalizes temporally extended subpolicies that the agent can choose from. Each option represents a sequence of low-level actions that are executed until a termination condition is met [26]. The option-critic architecture extends this concept by allowing discovering options rather than using pre-specified option policies [6]. Contrary to Feudal Networks, here the high-level controller is trained with gradients coming directly from the worker and no intrinsic reward is used. Hierarchical Reinforcement Learning with Off-Policy Correction

(HIRO), introduced in [20], enhances sample efficiency while maintaining high performance. To achieve that, an off-policy correction is integrated into the hierarchical model, more data-efficient when compared to on-policy methods. Another significant distinction between their method and prior approaches is having the goals encoded as a state observation in their raw form, eliminating the need for goal representation training.

Safe Hierarchical Reinforcement Learning. The core idea of our paper is to integrate safety mechanisms into the HRL architecture. Literature covering this problem already exists, although limited. In [30], the authors propose a two-level HRL method with the high-level agent consisting of an adaptive path planner based on A* and RRT*, able to generate safe and efficient paths, and the low-level ensures runtime safety with a Lyapunov-based safety monitor. In [16] a different approach is taken by integrating a high-level RL method which learns how to select the best alternative among non-learning safe low-level controllers. In [11], the Hierarchical Program Triggered Reinforcement Learning (HPRL) is proposed to solve the problem of safe navigation control. In HPRL, a structured program is responsible to trigger motion primitives that are trained using RL policies. The structured program is embedded with rule-based safety specifications and formal verification is used to ensure safety. Our model, in contrast, requires no handcrafted models and is composed of model-free agents and constraint models trained in an end-to-end fashion.

3 Preliminaries

3.1 Reinforcement Learning

In RL, we consider an agent that sequentially interacts with an environment modeled as a Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, R, P \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function which describes the system dynamics, where $P(s_{t+1}|s_t, a_t)$ is the probability of transitioning to state s_{t+1} given that the previous state was s_t and the agent took the action a_t. At each time step the agent observes the current state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$, transitions to the next state s_{t+1} drawn from the distribution $P(s_t, a_t)$, and receives a reward $R_t = R(s_t, a_t, s_{t+1})$.

3.2 Constrained Markov Decision Process

The Constrained Markov Decision Process (CMDP) extends the MDP framework by adding a set of K constraints, $c_i(s_t, a_t),\ \forall i \in [K]$ bounded by a set of thresholds $h_i,\ \forall i \in [K]$. The goal of the RL agent is to find the policy that maximizes the expected discounted return while keeping all constraints bounded by their safe thresholds:

$$\begin{aligned} &\max_{\theta} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \mu_\theta\left(s_t\right)\right)\right] \\ &\text{s.t. } c_i(s_t, \mu_\theta(s_t)) \leq h_i \ \ \forall i \in [K], \end{aligned} \tag{1}$$

where γ is the discount factor and $\mu_\theta : \mathcal{S} \mapsto \mathcal{A}$ is the policy that maps states to actions, parametrized by the vector of weights θ.

To simplify the notation, the index i will be dropped, since we will mostly consider problems with a single constraint function.

3.3 Goal-conditioned Hierarchical Reinforcement Learning

Goal-conditioned HRL methods extend the MDP formulation above by adding a set of goals $\mathcal{G}$, i.e., $\mathcal{M} := \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, R, P \rangle$. We will follow the HIRO formulation introduced in [20], which consists of two hierarchies: a high-level controller with policy μ_{θ_h} and a low-level with policy μ_{θ_l}, parameterized by neural networks with parameters θ_h and θ_l as function approximators. The high-level controller aims to maximize the extrinsic reward R_t and generates a high-level action $g_t \sim \mu_{\theta_h}(s_t) \in \mathcal{G}$ every k steps, granting temporal abstraction. The lower-level policy observes the state s_t and goal g_t and outputs a low-level atomic action $a_t \sim \mu_{\theta_l}(s_t, g_t)$, which is applied to the environment. The higher-level controller provides the lower-level with an intrinsic reward $r_t = r(s_t, g_t, a_t, s_{t+1})$, which rewards the low-level controller when reaching the goal g_t.

4 Methodology

In this section, we present our take on making goal-conditioned HRL architectures safer, by adding a safety layer to filter unsafe actions in a two-level hierarchical model to keep a safety signal under a predefined threshold. This approach allows us to strike a balance between performance optimization and safety considerations. The overall architecture is shown in fig. 1, with the main building blocks to be detailed next.

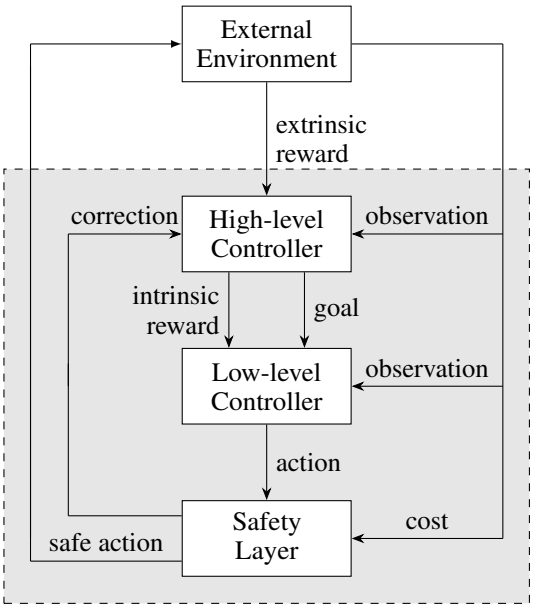


Figure 1: Safe HRL proposed architecture.

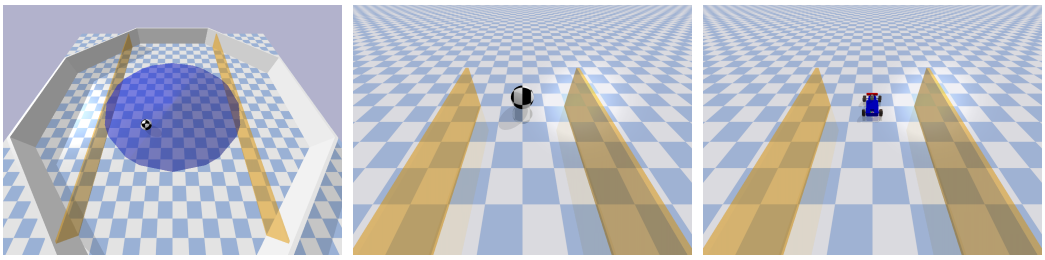
4.1 Safe Hierarchical Reinforcement Learning

At the core of our approach is the two-level hierarchy, consisting of a manager that chooses high-level goals and a low-level controller. The high-level policy is responsible to choose the goals that will return the most long-term cumulative reward based on the state observation. High-level actions might be encoded in different ways depending on the application, but for the locomotion tasks we define the goals as intermediate 2D coordinates, i.e., $q_{goal} = [x_g, y_g]$ (or a 3D coordinate if considering 3D locomotion, where $q_{goal} = [x_g, y_g, z_g]$). Therefore, the task of the high-level agent is to propose goal positions that will guide the agent's navigation to reach the positions necessary to fulfill the task (e.g., reach a goal position, pick an object, avoid an obstacle), analogous to a path planning algorithm.

To introduce safety into the framework, we incorporate a safety layer as proposed in [8]. This safety layer projects the actions into a safe set using linearized cost models approximated by neural networks. Although the high-level decisions could also be monitored, only the low-level actions have a direct influence on the constraint function shown in eq. (1). To make the high-level agent safety aware, a feedback signal with the safety layer's action corrections was added, which will be detailed next. The result is a closed-loop system that continuously monitors and adjusts the actions to maintain safety while pursuing goals that are effective in solving the task and keeping safety intervention at a minimum.

4.2 Safety Layer

Different authors proposed action projection into a safe set to achieve constraint satisfaction in RL systems. We follow the approach described in [8] and, similarly, in [22]. It consists of training a model f_w, parametrized by w, the underlying dynamics that rule the evolution of the cost function over time. With this model, and based on the current cost value c_t and the action selected by the agent a_t, it is possible to predict what the next step's cost value, c_{t+1} will be. We will also use the



(a) Safety Ball Circle. (b) Safety Ball Run. (c) Safety Car Run.

Figure 2: Testing environments.

simplification of approximating the cost function by the first-order Taylor expansion with respect to the action a_t:

$$c_{t+1} = c_t + f_w(s_t)^\top \cdot a_t, \tag{2}$$

where $\cdot$ represents the vector dot product.

The model f_w is pre-trained in a supervised manner with samples (s_t, a_t, c_t, c_{t+1}) experienced through environment interaction. With the cost model available, a safe action a_t^* can be calculated using the following constraint optimization problem:

$$\begin{aligned} a_t^* = \ & \arg\min_x \frac{1}{2}\|x - a_t\|^2 \\ & \text{s.t. } c_t + f_w(s_t) \cdot a_t^\top \leq h. \end{aligned} \tag{3}$$

Solving this optimization problem can be done using convex optimization solvers or integrating convex optimization layers to the policy model, as proposed in [3]. For problems with a single constraint active at a time, a closed for solution using the Lagrange multiplier can also be used.

4.3 High-level controller

The high-level controller is equivalent to what is seen in other goal-conditioned HRL methods, with the addition of another feedback signal that is the action correction given by the safety layer. This correction is calculated as the difference between the low-level controller action and the safe action, accounting for actions of multiple dimensions using the Euclidean norm, i.e., $\text{correction}_t = ||a_t - a_t^*||$. By introducing this correction, we aim to enable the high-level model to learn how to choose high-level actions that will incur in less safety-layer activations while optimizing the extrinsic reward, to mitigate negative impacts on performance caused by constraining the actions in detriment of safety.

Different options could be taken for enforcing the agent to minimize the action correction, but we opted to use reward shaping to punish the agent for high correction values. This is an effective alternative that requires no modification on the policy model or the training algorithm. A weight factor η was added to adjust the importance of the correction in the total shaped reward. The adjusted reward R_t^* is given by the equation below.

$$R_t^* = R_t - \eta \, \text{correction}_t. \tag{4}$$

5 Experiments

We conducted our experiments using the Bullet-Safety-Gym suite [15], an open-source benchmark suite inspired by the Safety-Gym benchmark [23]. The Bullet-Safety-Gym leverages the Bullet physics engine as its backend and is specifically designed to evaluate the performance of RL algorithms in constrained environments. Our experimental evaluation involved three distinct environments: (i) Safety Ball Circle, (ii) Safety Ball Run, and (iii) Safety Car Run.

The Safety Circle tasks, as introduced in [2], require agents to navigate along a circular path in a clockwise direction. The agent's reward is densely distributed and increases based on its velocity and proximity to the circle's boundary. In the Safety Run tasks, the agent's objective is to navigate through a corridor to reach a goal positioned at the opposite end. The reward in these tasks is proportional to the agent's progress towards the goal along the corridor. Both the Safety Circle and Safety Run environments incorporate safety zones bounded by two yellow walls. It is important to note that these walls are not rigid, allowing the robots to traverse them, and are located at fixed positions.

In our experiments, we employed two types of agents: the Ball agent, featured in environments (i) and (ii), and the Car agent, which offers a more complex representation of a mobile robot. The Ball agent simulates a rolling ball controlled by a two-dimensional force vector. On the other hand, the Car agent is designed as a simplified version of the MIT RaceCar model, allowing actuation over linear velocity and steering angle [1].

In order to integrate the safety layer, the original discrete cost functions provided by the benchmark were transformed into continuous functions. This modification was necessary due to the convex optimization techniques used for calculating safe actions. The revised cost function incorporates the tan^{-1} operator, which takes into account the distance between the agent and the closest wall boundary, denoted as $\hat{q}_{wall}$. Specifically, the cost at time step t is defined as $C_t = tan^{-1}(q_t - \hat{q}_{wall})$. When the agent's state lies within the safe region, the cost function returns negative values. However, if the agent crosses the safety limits, positive costs are assigned.

The experimental comparison will involve four different algorithms, comprising a goal-conditioned hierarchical reinforcement learning (HRL) approach an comparable flat policy and their safe variants integrating the safety layer. On the HRL side, we used the HIRO algorithm. The results include the vanilla HIRO, as presented in the original paper, as well as the safe HIRO variant that integrates our proposed safe architecture to the HIRO model. For a comparison with flat policies, we will include the TRPO algorithm [24], as well as safe-TRPO, which incorporates a safety layer to ensure constraint satisfaction, as demonstrated in [23].

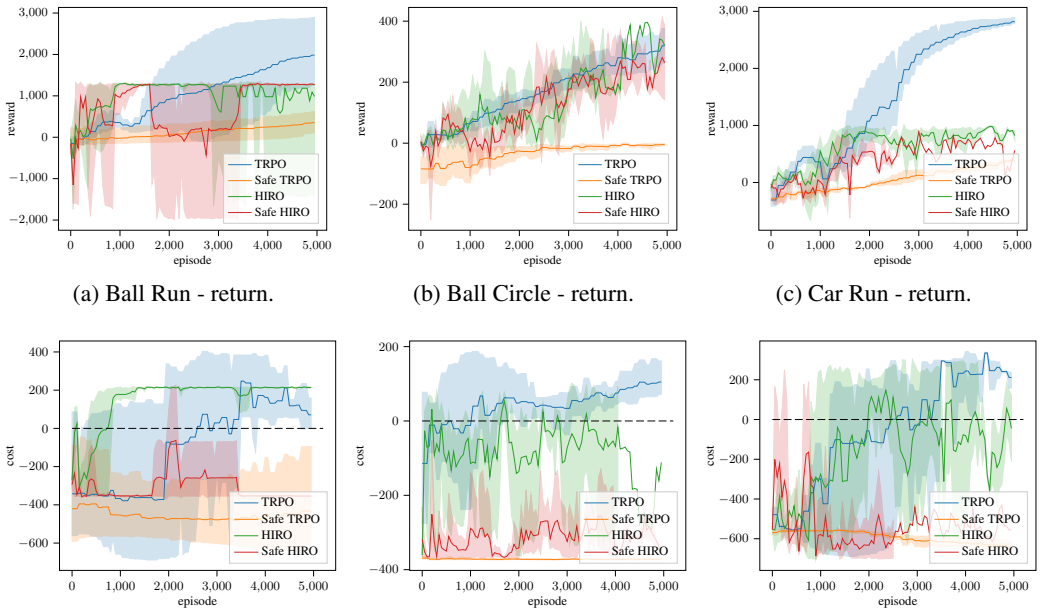
5.1 Learning the constraint functions

The constraint functions are approximated by neural networks trained with data sampled from the environment, allowing the method to scale to complex and high-dimensional problems. Like most DNNs trained in a supervised learning manner, training the constraint models might require lots of data, ranging from a few thousand samples for simple models up to millions when considering larger DNNs and more complex data representations (as an example., in [21] $50 \cdot 10^6$ data samples are used to train a model predictive control (MPC) model for robot manipulation tasks). When simulation environments are available and acquiring data can be done at a low cost, a suitable approach is to let an agent explore the environment to acquire the necessary data, either by taking random actions or following some other exploratory policy. Collecting data in the real world is more complicated since one cannot simply let the agent explore taking random actions without compromising safety. Using imitation learning (e.g., collecting data from human operators), or letting a safe and verifiable controller control the system while the data is being collected are possible approaches. If acquiring the data is too expensive or difficult, it is important to consider if designing the constraint models using other means aside from learning them might be a more suitable approach.

5.2 Hyperparameters

The constraint models are fully connected networks composed of two hidden layers with dimensions of [64, 64] respectively. A dropout layer was added to minimize the risk of 500,000 data points were randomly sampled to train each constraint model offline. The training was done with a learning rate of $1 \cdot 10^{-3}$. Each model was trained for 10 epochs with a batch size of 512.

The HIRO agent is composed of two TD3 agents, one for each controller in the hierarchy, and the hyperparameters follow what is shown in [10]: For both high- and low-level agents, an actor learning rate of $1 \cdot 10^{-4}$ and a critic learning rate of $1 \cdot 10^{-3}$ were used. The policy noise parameter σ of 0.2 is used, i.e., $\mathcal{N}(0, 0.2)$ is added to the actions, with the noise level being clipped to (-0.5, 0.5). The delayed policy updates happen every d = 2 iterations. The target networks are updated with τ = 0.005. A discount factor γ of 0.99 was used. Finally, $\eta = 0.1$ was set to the high-level reward correction weight.



(a) Ball Run - return. (b) Ball Circle - return. (c) Car Run - return.

(d) Ball Run - cost. (e) Ball Circle - cost. (f) Car Run - cost.

Figure 3: Results for the simulation environments. Comparison between the vanilla HIRO, safe-HIRO, TRPO, and safe-TRPO methods. The results regarding the episodic cumulative rewards can be seen in the first row (figs. 3a to 3c) while the episodic cumulative costs values are shown in the row below (figs. 3d to 3f.)

The results, shown in fig. 3, are averaged over 3 successful runs obtained with distinct seeds.

5.3 Results

The results obtained in the Ball Run environment are shown in figs. 3a and 3d. We can see that for this environment both constrained methods (safe-HIRO and safe-TRPO) can keep the system inside the safe region, i.e., with the cost values negative. By analyzing the safe-TRPO curves it becomes clear that the performance drops drastically when constraining the TRPO model. Safe-HIRO presents a few spikes in the cost values from episodes 2000 and 3000 which is probably caused by errors in the constraint model since a perfect model would always give a safe action able to avoid the system to cross the safety boundaries. Overall, the system can retain the same performance as HIRO (slightly above 1,000 of return) while satisfying the safety constraints.

Similar results were obtained in the Ball Circle environment, as shown in figs. 3b and 3e. In this plot, it becomes even more evident the difference in stability while learning. TRPO is very consistent and stable on the learning curves, while the HRL models displayed greater variation in reward over time. Despite the noisy results, the safe-HIRO accompanies HIRO in the reward curve while effectively managing the associated costs. On the other hand, safe-TRPO's performance is not even comparable to any of the other methods, despite having a better cost reduction and more stable curves than safe-HIRO.

For the Car Run environment, presented in figs. 3c and 3f, The TRPO model exhibited remarkable results with its strong performance while both HRL models perform relatively poorly when compared to it. However, despite the lower overall performance, once more there is no significant degradation in the HRL models' performance when the safety layer is activated, leading to a significant improvement in terms of constraint satisfaction. On the other hand, the safe TRPO overall performance lags behind the other models, although successfully keeping the cost values low.

6 Limitations

As shown in the previous section, the proposed HRL architecture was successful in creating models that learn how to minimize the cost for the constraint functions while keeping a comparable performance to the unconstrained counterpart. The optimization of these constraint functions serves as a proxy for ensuring safety in the given examples. Nevertheless, it is essential to acknowledge that the system does have its limitations, which are outlined below:

One-step-ahead monitoring. Estimating how much impact the agent's actions will have in the future is important for both performance and safety concerns. Real-world results might differ substantially since some of the simplifications from the simulation environments are not present (e.g., absence of noise, unclipped actions). If we expect this method to be deployed in real robots, a larger constraint prediction horizon should be considered, similar to model predictive control (MPC) approaches, to allow the agent to safely plan ahead and account for physical limitations when controlling the system.

Constraint model uncertainties. The safety layer works by monitoring if the actions taken by the low-level controller will lead to a safety violation. Therefore, safety assurance is completely dependent on the ability of the constraint models to correctly learn the constraint dynamics and make assertive predictions. However, determining the reliability and robustness of DNN models is difficult and considered an open question for most applications. Integrating a good uncertainty estimation could help to mitigate this issue.

Increased number of parameters. The proposed hierarchical reinforcement learning (HRL) architecture introduces a more intricate structure compared to a flat policy, resulting in an increased number of parameters. As a consequence, a higher number of hyperparameters have to be tuned. Moreover, we observed that finding a parameter configuration for achieving convergence in the HRL agent is notably more challenging compared to its flat policy counterpart, which is most likely related to the higher complexity.

7 Conclusion and Future Work

In this paper, we showed that flat RL policies can suffer from performance loss when methods to enforce safety constraints are used and that a hierarchical model proved less susceptible to this degradation in our test scenarios. This conclusion comes from results obtained in simulation experiments with constrained navigation tasks. The proposed constrained HRL architecture is simple, as shown in fig. 1, and scalable since both constraint models and RL agents use deep neural networks as function approximators.

By implementing a strategy to penalize the high-level controller when action corrections are required, the system was able to learn how to generate goals that minimize intervention from the low-level policy. The introduction of this penalty mechanism enabled the hierarchical architecture to exhibit improved adaptability and reduced reliance on corrective actions. TRPO, on the other hand, is affected to a greater extent, as its policy is represented by a single end-to-end model that lacks the flexibility to make decisions at various levels of abstraction.

This work opens up exciting avenues for further research and investigation. Ensuring safety while maintaining a satisfactory functionality level is essential to solving safety-critical problems with artificial intelligence and the obtained results address this issue successfully. However, the method is not perfect and the main limitations were also outlined. One important direction to explore is the integration of larger prediction horizons. As we strive to tackle real-world problems, extending the planning and prediction capabilities to react to situations far in the future becomes crucial. Dealing with model uncertainties is also an open point for inspection. Real-world environments often exhibit complex dynamics and inherent uncertainties, which can pose challenges to safe decision-making. Developing techniques to handle and mitigate these uncertainties within the hierarchical framework would enhance the robustness and reliability of the agent's behavior. Furthermore, extending the safe hierarchical framework to multi-agent scenarios presents an exciting topic for future inspection. In such settings, each agent has its own decision-making process, and ensuring safety becomes a localized concern. Developing approaches that enable agents to communicate, coordinate, and reason about safety within a collaborative framework would be valuable for real-world applications involving multiple interacting agents.

Acknowledgments This work was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy as part of a project to support the thematic development of the Institute for Cognitive Systems.

References

[1] Mit racecar mobile platform, 07 2022.

[2] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[3] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[5] E. Altman. *Constrained Markov decision processes*. PhD thesis, INRIA, 1995.

[6] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[7] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[8] 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.

[9] P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[10] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[11] B. Gangopadhyay, H. Soora, and P. Dasgupta. Hierarchical program-triggered reinforcement learning agents for automated driving. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[12] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[13] C. Gehring and D. Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1037–1044, 2013.

[14] A. Geramifard, J. Redding, and J. P. How. Intelligent cooperative control architecture: a framework for performance improvement using safe learning. *Journal of Intelligent & Robotic Systems*, 72:83–103, 2013.

[15] S. Gronauer. Bullet-safety-gym: A framework for constrained reinforcement learning. 2022.

[16] J. Li, L. Sun, J. Chen, M. Tomizuka, and W. Zhan. A safe hierarchical planning framework for complex driving scenarios based on reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2660–2666. IEEE, 2021.

[17] B. Lütjens, M. Everett, and J. P. How. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, pages 1328–1337. PMLR, 2020.

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

[19] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes, 2012.

[20] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[21] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe. Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5(2):3050–3057, 2020.

[22] T.-H. Pham, G. De Magistris, and R. Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.

[23] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[26] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[27] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[28] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2, 2019.

[29] A. Wachi and Y. Sui. Safe reinforcement learning in constrained markov decision processes. In *International Conference on Machine Learning*, pages 9797–9806. PMLR, 2020.

[30] Z. Xiong, I. Agarwal, and S. Jagannathan. Hisarl: A hierarchical framework for safe reinforcement learning. In *SafeAI@ AAAI*, 2022.

[31] K. Zhu and T. Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.