BULLET-SAFETY-GYM: A FRAMEWORK FOR CONSTRAINED REINFORCEMENT LEARNING

TECHNICAL REPORT

Sven Gronauer
Department of Electrical and Computer Engineering
Technical University of Munich
Arcisstr. 21, 80333 Munich, Germany
sven.gronauer@tum.de

January 24, 2022

ABSTRACT

The *Bullet-Safety-Gym* is an open-source framework to train and assess safety specifications in constrained reinforcement learning problems. We have implemented 16 environments differing in complexity and design. The framework is entirely written in Python and builds open the freely available PyBullet physics engine. Based on our environments, we evaluate five state-of-the-art policy gradient algorithms and provide results as a baseline for future research. To this end, we discuss our findings and outline possible limitations of the investigated algorithms. Our framework is available at: https://github.com/SvenGronauer/Bullet-Safety-Gym

1 INTRODUCTION

The programming of intelligent control strategies a-priori is a demanding task for complex robot systems. When requirements such as safety or robustness must be satisfied, the controller design becomes additionally complicated. Reinforcement learning (RL) enables the automated development of control strategies through a data-driven approach instead of explicitly designing such. In recent years, the field of RL has been driven by outstanding successes and raised a surge of interest in controlling robot systems through a trial-and-error approach. The combination of RL and deep learning methods excel at problems that can be quickly simulated like robotics [23, 31] and video games [24, 38], or where an exact model is known but long-term planning is computationally not tractable, e.g. board games like Go [32].

However, outside the simulator RL faces several challenges. When applied to a real-world robot system, the agent usually cannot afford to learn from scratch due to the expense of data. Also, failure is undesirable since it may break the robot or harm its environments, which strongly promotes the embedding of safety specifications into the learning system. The increasing computational power and availability of hardware resources have made simulations the paramount approach for building and testing data-driven control policies. In the simulation, failure is acceptable and even desirable to learn from bad outcomes. Thus, simulation environments are often used to pre-train a well-behaving control strategy before transferring it to the real-world system [20, 27, 36]. For sim-to-real transfer approaches, it may be beneficial to include safety specifications already during the training in simulation.

In this report, we propose an open-source framework called *Bullet-Safety-Gym* that allows incorporating safety specifications into the RL training. The focus is deliberately on meeting the safety specifications at the end of training which can be of practical importance for a sim-to-real transfer. With this work, we want to address two points. First, our framework can be used to train and assess agents with safety specifications in RL problems. We collect environments that were proposed in recent works and merge them into one unified and standardized code-base. This eases the comparability between research experiments and promotes the reproducibility of results. Second, our environments can suit as a benchmark to quantify the impact of new algorithm proposals. To provide a baseline, we conducted experiments with five state-of-the-art policy gradient algorithms and evaluated the performance based on a constrained cost criterion. Although our framework also yields the possibility to investigate safe exploration during training, this

lies outside the scope of this work. To this end, we discuss possible limitations of the current practices in the domain of constrained RL and suggest possible future work.

2 Related Work

Safety. Different notions of safety exist within the artificial intelligence community [3, 26, 28]. One definition tailored to the RL setting describes safety in terms of the maximization of some performance measure where it is mandatory to ensure system performance and respect safety constraints throughout learning and at deployment [17]. A similar view on safety is the prevention of error states from which the original state of the system cannot be recovered. Error states can be defined a-priori by constraints [2, 12, 35] and are tightly coupled to the concepts of reach-ability [25] and stability [6]. Although no consistent definition of safety exists within the community, methodologies about the assessment of safety have been broadly discussed in the literature [8, 17, 26, 28].

Simulation Frameworks. Simulators have established themselves as the de-facto standard to develop control policies with RL methodologies. Carried by the rapid development of new algorithm proposals, a plethora of frameworks and benchmarks has emerged. The most known is *Gym* [7] which defines the common programming interface for the interaction between the learning agent and the environment. Video games are popular test-beds for high-dimensional sensory input data, e.g. the Atari games for 2D learning tasks [5], the DeepMind Lab [4] for 3D learning environments, or the StarCraft learning environment [**Vinyals2017StarCraftEnv**] for multi-agent settings. Besides video games, commonly used benchmarks are grounded on physics simulators. Frameworks have been proposed for robotic learning with continuous control for diverse tasks [34], imitation- and active-learning [13], manipulation tasks [16], or robot-human interaction and physical health-care [15]. Most similar to our work is [28] who provide a benchmark for safe RL based on the MuJoCo simulator [37]. Our work includes tasks from [28] and collects additional environments used to benchmark new algorithm proposals, e.g from [1, 9] and merges them into one framework based on the PyBullet physics engine [11]. While [28] considered safe exploration, our focus lies on the safety at convergence, where agents are obliged to satisfy the constraints only at the end of training but the exploration may be unsafe.

3 Preliminaries

A Markov Decision Process (MDP) is formalized by a tuple $(\mathbb{X}, \mathbb{U}, \mathscr{P}, r, \mu)$, where $\mathbb{X}$ and $\mathbb{U}$ denote the state and action space, respectively. $\mathscr{P} : \mathbb{X} \times \mathbb{U} \to P(\mathbb{X})$ describes the state transition probability, $r : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ is the reward function, and μ denotes the initial state distribution. The goal of the agent is to learn a control policy $\pi : \mathbb{X} \to P(\mathbb{U})$ that maximizes the expected return $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} r(x_t, u_t) \mid x_0\right]$ under the finite-horizon T. We use $\tau \sim \pi$ as a shortcut for the data collected under the policy π, i.e. $x_{t+1} \sim \mathscr{P}(\cdot|x_t, u_t)$, $u_t \sim \pi(\cdot|x_t)$, and $x_0 \sim \mu$. We denote the policy as π_θ when it is parametrized by a vector θ.

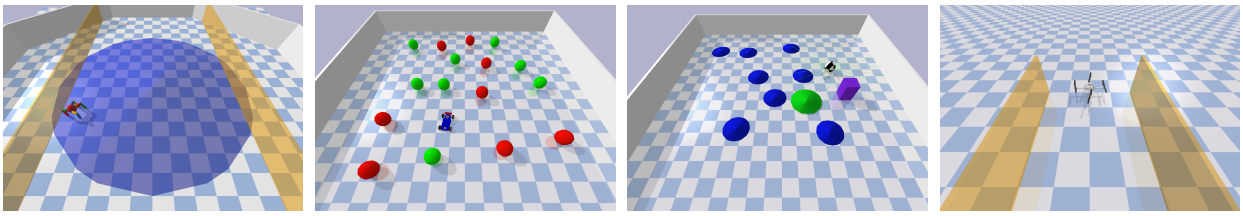
The Constrained Markov Decision Process (CMDP) [2] extends the MDP formalism with an auxiliary cost function $c : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$. While opting for reward maximization, agents are obliged to fulfill cost constraints which are expressed by the inequality $H \leq d$. We denote the cost limit as $d \in \mathbb{R}$ and the expected cost under policy π_θ as $H(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} c(x_t, u_t) \mid x_0\right]$. Therefore, in the setting of CMDPs, we seek to find the policy parameter vector θ^* over the set of feasible policies $\Pi_C = \{\pi_\theta \in \Pi_\theta \mid H \leq d\}$ that maximizes J, i.e. $\pi_{\theta^*} = \arg\max_{\pi_\theta \in \Pi_C} J$.

4 Bullet-Safety-Gym

Although many benchmark environments have been proposed in recent years [14, 34], only a minority regards safety aspects [28]. Reproducibility and comparability are key ingredients for scientific advances, making it desirable to have free and open-source software which may eventually accelerate the research progress by making RL methods available to a broader community.[1]

In this section, we describe the proposed *Bullet-Safety-Gym* framework which contains a diverse range of RL tasks to that incorporate safety specifications. Although an abundance of new algorithm proposals has emerged in recent years [1, 33, 39, 40], many works test only a tailored set of environments. There is not yet a consistent use of benchmark tasks, which hampers comparability and complicates the assessment of new scientific contributions. In the remainder of this section, we elaborate on the design principles behind our environments and introduce the agents and tasks that were implemented.

[1] The original motivation behind the *Bullet-Safety-Gym* was that the majority of frameworks depended on proprietary software like MuJoCo. However, due to the acquisition of MuJoCo by DeepMind and the announcement that MuJoCo will be released in 2022 with open source code, the circumstances have luckily changed and most frameworks are now free to use.



(a) Circle (b) Gather (c) Reach (d) Run

Figure 1: The *Bullet-Safety-Gym* provides four agents with different complexity and locomotion behaviors and is shipped with four pre-defined tasks that differ in reward and cost design. In total, our framework contains 16 environments.

4.1 Environment Design

The implementation of our proposed environments follows three design principles: standardization, modularization, and generalization. We integrated the *Bullet-Safety-Gym* into the standardized interface of Gym [7] where the interaction between agent and environment is modeled in stages. The agent perceives observations from its environment, acts upon them, and the environment responds with information including the follow-up state, the reward and a cost signal. To promote easy customization and a high degree of extendability, our framework has a modular structure where the base class `EnvironmentBuilder` organizes the creation of the world in the physics simulator with the desired configuration of the agent, task, and obstacles. Also, agents and tasks are modular and can be changed arbitrarily. The layout of the world can be freely adapted, e.g. which and how many obstacles are spawned and how they interact with their surroundings through predefined moving patterns. The last design principle of our environments is the randomization of the world layout. While the number and types of obstacles as well as the agent stay constant, the state of the world is reset and shuffled at the beginning of new episodes. Layout randomization helps to avoid over-fitting and encourages agents to generalize over different world settings [10]. Further, we aim to prevent the vulnerability towards distributional shift, which is one of the major concerns for safe and robust artificial intelligence [3].

4.2 Agents

We focus on applications with continuous state and action spaces that primarily involve the learning of locomotion and gait behaviors. We implemented four agents that differ in complexity and dynamics.

- The spherical shaped agent called *Ball* $\left(\mathbb{X} \subset \mathbb{R}^7, \mathbb{U} \subset \mathbb{R}^2\right)$ can freely move on the ground plane and is controlled by a two-dimensional force vector.
- *Car* is a four-wheeled agent $\left(\mathbb{X} \subset \mathbb{R}^7, \mathbb{U} \subset \mathbb{R}^2\right)$ based on the *MIT RaceCar* with a simplified control scheme consisting of the target wheel velocity for all wheels and the target steering angle.
- The air vehicle called *Drone* $\left(\mathbb{X} \subset \mathbb{R}^{17}, \mathbb{U} \subset \mathbb{R}^4\right)$ is based on the *AscTec Hummingbird* quadrotor. The robot is controlled by setting the desired velocity for each rotor.
- *Ant* is a quadrupedal agent $\left(\mathbb{X} \subset \mathbb{R}^{33}, \mathbb{U} \subset \mathbb{R}^8\right)$ composed of nine rigid bodies, including a torso and four legs. Each leg consists of two actuators which are controlled torque-based.

Each agent can fully observe its own body state space. However, some tasks require the sensing of the environment for obstacles. In such cases, the agent is equipped with external sensors that cast laser rays $\left(\mathbb{X} \subset \mathbb{R}^{24}\right)$ to detect nearby obstacles. Task-specific information may also be needed such as the distance to a goal and are additionally added to the agent's observations.

4.3 Tasks

We deliver the *Bullet-Safety-Gym* with the four tasks illustrated in Fig. 1. In total, we introduce 16 environments with different cost and reward designs.

- *Circle*. The agent is expected to move on a circle in clock-wise direction [1]. The reward is dense and increases by the agent's velocity and by the proximity towards the boundary of the circle. Costs are received when an agent leaves the safety zone defined by the two yellow boundaries.
- *Gather*. Agents are supposed to navigate and collect as many green apples as possible while avoiding red bombs [14]. In contrast to the other tasks, the agent receives only sparse rewards in the Gather task when reaching apples. Costs are also sparse and are received when touching bombs [1].

- *Reach*. Agents are expected to move towards a series of goals [28]. When the agent enters the goal zone, the goal is re-spawned such that the agent has to reach the next position. Obstacles are placed to hinder the agent from trivially finding solutions. We implemented obstacles with a physical body, into which agents can collide and receive costs, and ones without collision shape that produce costs for touching. Rewards consist of a dense component for moving closer to the goal and a sparse component for entering the goal zone.
- *Run*. The agent is rewarded for running through an avenue between two safety boundaries [9]. The boundaries are non-physical bodies that can be penetrated without collision but provide costs. Additional costs are received when exceeding an agent-specific velocity threshold. A characteristic of the Run tasks is that policies can quickly overshoot the cost constraint after a few policy iterations, which requires algorithms to perform large cost reduction steps already at the early stage of training.

5 METHODS

In this section, we explain the methodology applied to benchmark our proposed environments. We introduce the metrics to measure and assess safety specifications and enumerate five algorithms, which may suit as a benchmark baseline to evaluate future algorithm proposals.

We consider a constrained criterion where we want to maximize the expected utility of a policy that also fulfills cost constraints lower than some given bound $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} r(x_t,u_t)\right]$ *s.t.* $H(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} c(x_t,u_t)\right] \leq d$. We train agents for N epochs after which we evaluate the policy deterministically without exploration noise. The safety of a policy is measured at convergence, i.e. we neglect cost violations during the training as long as the learned policy can satisfy the constraints at the end of training. This is a valid assumption for transfer learning scenarios where it is crucial to obtain an initially safe policy from the simulator before the policy is deployed on a real-world system.

Akin to the metrics used in [28], we consider algorithm A better than B when $J(A) > J(B)$ subject to $H(A) \leq d$ and $H(B) \leq d$. Further, A outperforms B when B is not able to satisfy the cost constraints, i.e. $H(A) \leq d < H(B)$. For an equitable comparison between algorithms, we select the best average performance obtained over a hyper-parameter grid search.

In this paper, we consider on-policy policy gradient methods, which generate a batch of trajectories based on the current policy and then use this data to determine a suitable direction for the policy update. In particular, we compare the following five algorithms:

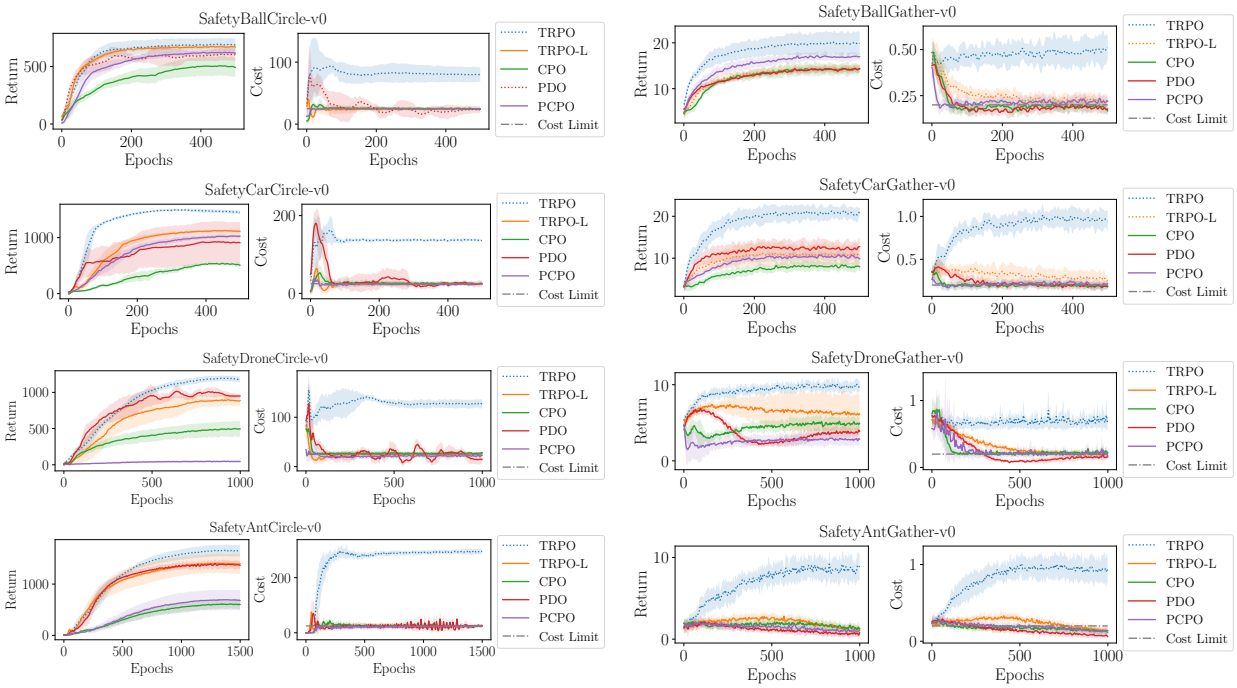
- *Trust-Region Policy Optimization (TRPO)* [31] is an un-constrained algorithm which does not regard cost signals. Each policy update step lies within a pre-defined trust-region size.
- *TRPO-L* applies a Lagrangian relaxation to the TRPO objective, transforming the constraint problem to an equivalent unconstrained one which accounts for cost violations. The Lagrange multiplier is a learnable and stateful variable to trade-off rewards and costs.
- *Constrained Policy Optimization (CPO)* [1] optimizes the trust-region problem and determines the Lagrange multiplier for constraint satisfaction from scratch at each policy update step. We use a simplified version without cost shaping similarly to [28].
- *Primal-dual Optimization (PDO)* uses as a learnable and stateful Lagrange multiplier that is used to trade off reward and cost gradients in a trust-region update step. Our implementation follows [1].
- *Projection-based Constrained Policy Optimization (PCPO)* is an extension to CPO that optimizes the policy objective in two steps. The first step is an unconstrained parameter update while the second step regards constraint violations by projecting the policy back onto the constraint set [39].

We use TRPO as an un-constrained algorithm to estimate the upper bound of the achievable returns when safety aspects are not regarded. Thus, the obtained results in Sect. 6 can be better assessed and increase comparability between the four constrained algorithms. To maintain a clear separation between environments and algorithms, we implemented algorithms in a separate code-base[2].

6 EXPERIMENTS

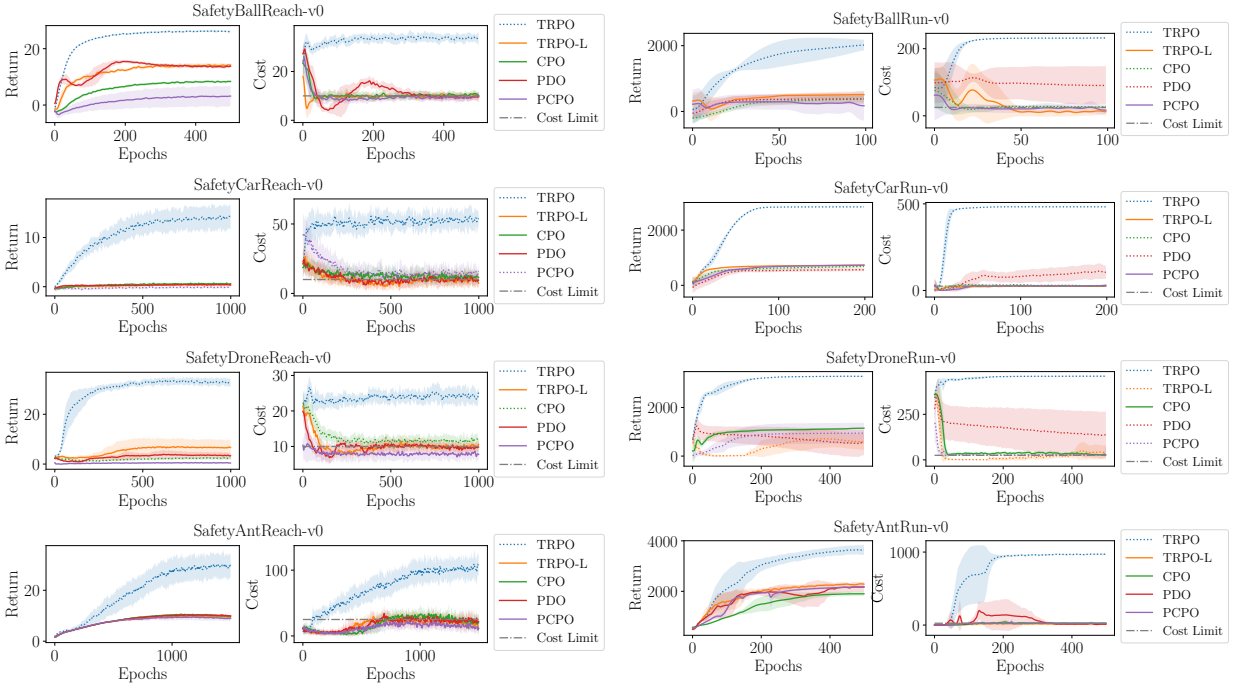
We aligned the hyper-parameters for the experiments to the ones discussed in [19] and provide an overview table in the Appendix. We applied a distributed learner setup where the policy gradients were computed and averaged across

[2] https://github.com/SvenGronauer/RL-Safety-Algorithms



(a) Circle

(b) Gather



(c) Reach

(d) Run

Figure 2: The learning curves show the best hyper-parameter setting of the five benchmarked algorithms in each introduced environment. Thick lines denote that an algorithm was able to fulfill the cost constraint on average whereas dotted lines indicate no constraint satisfaction at the end of the training. Each algorithm was averaged over four independent random seeds and the shaded areas show the standard deviation.

Table 1: Performance overview of benchmarked algorithms in the *Bullet-Safety-Gym* environments. The scores are normalized to the return J_{TRPO} achieved by TRPO and H is set in relation to the environment-specific cost limit d. The constrained algorithm with the best score is highlighted with bold font.

Algorithm	TRPO	TRPO-L	CPO	PDO	PCPO
Env	$J/J_{TRPO}\,(H/d)$	$J/J_{TRPO}\,(H/d)$	$J/J_{TRPO}\,(H/d)$	$J/J_{TRPO}\,(H/d)$	$J/J_{TRPO}\,(H/d)$
BallCircle	1.00 (3.19)	**0.96 (0.97)**	0.72 (0.99)	0.87 (1.03)	0.89 (0.93)
CarCircle	1.00 (5.49)	**0.76 (0.89)**	0.34 (0.97)	0.63 (0.91)	0.71 (0.93)
DroneCircle	1.00 (5.10)	0.75 (0.93)	0.43 (0.99)	**0.80 (0.56)**	0.05 (0.93)
AntCircle	1.00 (11.8)	**0.74 (0.97)**	0.33 (0.98)	0.73 (0.92)	0.42 (1.00)
BallGather	1.00 (2.47)	0.75 (1.09)	0.71 (0.82)	0.75 (0.91)	**0.85 (0.98)**
CarGather	1.00 (4.78)	0.53 (1.35)	0.40 (0.82)	**0.59 (0.98)**	0.51 (0.96)
DroneGather	1.00 (3.20)	**0.63 (0.71)**	0.51 (0.92)	0.38 (0.81)	0.29 (0.85)
AntGather	1.00 (4.54)	0.11 (0.82)	**0.14 (0.57)**	0.08 (0.37)	0.08 (0.60)
BallReach	1.00 (3.43)	**0.55 (0.81)**	0.32 (0.99)	0.52 (0.83)	0.11 (0.96)
CarReach	1.00 (5.66)	0.02 (0.78)	**0.04 (0.88)**	**0.04 (0.84)**	0.00 (1.14)
DroneReach	1.00 (2.63)	**0.21 (0.90)**	0.08 (1.21)	0.10 (0.90)	0.01 (0.84)
AntReach	1.00 (4.16)	**0.32 (0.94)**	**0.32 (0.46)**	**0.32 (0.97)**	0.29 (0.58)
BallRun	1.00 (9.28)	**0.25 (0.52)**	0.18 (1.05)	0.18 (3.58)	0.09 (0.30)
CarRun	1.00 (19.4)	**0.23 (0.87)**	0.21 (1.02)	0.17 (3.94)	0.23 (0.91)
DroneRun	1.00 (18.4)	0.18 (1.60)	**0.35 (0.96)**	0.18 (5.72)	0.29 (1.01)
AntRun	1.00 (38.9)	**0.63 (0.61)**	0.52 (0.98)	0.60 (0.48)	0.60 (0.98)

64 processes. We used as the discount factor $\gamma = 0.99$, collected batches of size $B = 32000$ for Ball and Car and $B = 64000$ for Drone and Ant agents. The number of epochs depended on the complexity of the task and was chosen from $N \in [100, 1500]$. As neural network architecture, we used the same structure for both policy and value networks, i.e. multi-layer perceptrons with two hidden layers consisting of 64 neurons each followed by tanh non-linearities. The weights were initialized with Kaiming Uniform and biases were set to zero vectors. The value networks were optimized with Adam [22]. The Lagrange multiplier was optimized by stochastic gradient descent (SGD) for TRPO-L and Adam for PDO. We used Generalized Advantage Estimation (GAE) [29] to reduce the variance of critic estimates with $\lambda = 0.95$ for rewards and $\lambda_c = 0.95$ for costs.

Over the training, we deployed a stochastic policy in form of a Gaussian $u \sim \mathcal{N}(\pi(x), \varepsilon I)$ with the identity I and the exploration noise $\varepsilon \in \mathbb{R}$ that was linearly annealed towards zero. For evaluation purposes, we deactivated the exploration noise and benchmarked the performance of the policy deterministically with respect to J and H.

6.1 Results

The learning curves of the tested algorithms are depicted in Fig. 2 and an overview of the scores can be taken from Table 1. For each algorithm, we searched over a grid of hyper-parameters and averaged the scores over four independent random runs. We picked the best return fulfilling the cost constraint $H \leq d$. If an algorithm could not fulfill the cost constraint over all hyper-parameter settings, the hyper-parameter setting with the lowest cost violation was selected. The values used for the grid search can be found in the Appendix.

We observed that TRPO-L produced strong results and outperformed CPO, PCPO and PDO in 10 out of 16 tasks. However, TRPO-L revealed a weaker performance in the Gather tasks where reward and cost signals are sparse, whereas the other three constrained algorithms were always able to steer the costs towards the desired limit. PDO struggled to reduce the costs in the Run tasks where policies quickly overshoot the cost constraint after a few training iterations, whereas CPO, PCPO and TRPO-L were able to perform cost reduction steps until constraint satisfaction in most cases.

Overall, we noticed that no hyper-parameter configuration consistently performed well in terms of cost satisfaction for all tasks. Each task had to be tuned individually via hyper-parameter grid search, which renders it difficult to provide good default hyper-parameter values in general. This observation elicits that the assessment of experimental results can be misleading when only one hyper-parameter configuration is evaluated.

6.2 Discussion

Although TRPO-L outperformed CPO, PCPO and PDO in most of our experiments, Lagrangian methods can be susceptible to oscillations around the cost limit. We found that oscillations intensify when replacing SGD with the Adam optimizer, rendering the behavior of Lagrangian relaxation methods strongly influenceable by the chosen

optimizer of the Lagrange variable. Oscillation behavior was also observed in [28] who tested Lagrangian relaxation variants of the Proximal Policy Optimization [30] and TRPO algorithm. The first research efforts were put into this issue in [33] who stabilized the learning by a proportional-integral-derivative controller.

We noticed in our experiments that the annealing of the exploration noise is crucial and fosters the satisfaction of the constraints towards the end of the training. CPO was observed in [28] to overshoot in costs when using a constant exploration noise factor. However, we were able to fulfill the cost constraint with CPO when using a linear exploration noise annealing - even without the additional cost shaping as it was applied in the original publication [1].

Besides the necessity to tune hyper-parameters for each environment individually, we observed a high susceptibility towards the chosen hyper-parameter values for all tested algorithms (see Appendix). Although this is a well-known phenomenon for deep RL in general [21], we found that this sensitivity issue is further exacerbated in the constrained RL setting due to the additional algorithmic components and extra hyper-parameters required for accounting costs. We also noticed that it was necessary to search over a broad range of hyper-parameter values in order to find a hyper-parameter combination that was able to satisfy the cost constraints averaged over all independent runs. Thus, we recommend that new algorithm proposals should be assessed over hyper-parameter grids in order to obtain an equitable performance comparison and not only use the default parameters.

A current limitation is that all tested algorithms struggle to consistently respect cost constraints over different random runs and hyper-parameter settings. Further, none of the investigated algorithms was able to produce zero cost violations throughout the whole training. This suggests that on-policy policy gradient algorithms may be unsuitable for applications where constraint satisfaction is required at all times, e.g. the safe exploration with real-world robots.

7 CONCLUSIONS

Benchmark frameworks are an integral component for assessing scientific progress, especially in domains where the pace of research is swift. Despite the increasing awareness for safety and the flourishing number of publications in the RL community, most works concerning safety conduct experiments on self-created and individual environments, aggravating the assessment of new contributions.

In this paper, we introduced the *Bullet-Safety-Gym* as a framework to train and assess safety aspects in constrained reinforcement learning problems. We collected and unified different environments that were proposed in recent works and provide an open-source Python package. In our experiments, we evaluated five algorithms based on the 16 environments of the *Bullet-Safety-Gym* which can serve as a baseline for future work.

Future work may comprise the investigation of off-policy algorithms and the extension to multi-dimensional cost signals which adds extra complexity to the training. Since we deliberately focused on safety satisfaction at the end of the training, it might be also of interest to investigate the safety aspects of the *Bullet-Safety-Gym* environments during the training. Lastly, another avenue for future research might be safe reinforcement learning in the multi-agent setting, which is still mostly unexplored as pointed out in [18].

ACKNOWLEDGMENT

I thank the Federal Ministry of Education and Research who sponsored this project under the funding code 01IS17049. Further, I want to acknowledge the support of Xavier Oliva and Pooya Kangani on parts of the implementation.

References

[1] Joshua Achiam et al. “Constrained Policy Optimization”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 22–31.

[2] Eitan Altman. *Constrained Markov decision processes*. CRC Press, 1999.

[3] Dario Amodei et al. “Concrete Problems in AI Safety”. In: *CoRR* abs/1606.06565 (2016).

[4] Charles Beattie et al. “DeepMind Lab”. In: *CoRR* abs/1612.03801 (2016).

[5] M. G. Bellemare et al. “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279.

[6] Felix Berkenkamp et al. “Safe Model-based Reinforcement Learning with Stability Guarantees”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 908–918.

[7] G. Brockman et al. “OpenAI Gym”. In: *ArXiv* abs/1606.01540 (2016).

[8] Lukas Brunke et al. "Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning". In: *CoRR* abs/2108.06266 (2021).

[9] Yinlam Chow et al. "Lyapunov-based Safe Policy Optimization for Continuous Control". In: *CoRR* abs/1901.10031 (2019).

[10] Karl Cobbe et al. "Quantifying Generalization in Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 1282–1289.

[11] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[12] Gal Dalal et al. "Safe Exploration in Continuous Action Spaces". In: *CoRR* abs/1801.08757 (2018).

[13] Brian Delhaisse, Leonel Rozo, and Darwin G. Caldwell. "PyRoboLearn: A Python Framework for Robot Learning Practitioners". In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 30 Oct–01 Nov 2020, pp. 1348–1358.

[14] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1329–1338.

[15] Zackory Erickson et al. "Assistive Gym: A Physics Simulation Framework for Assistive Robotics". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020).

[16] Linxi Fan et al. "SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, 29–31 Oct 2018, pp. 767–782.

[17] Javier García, Fern, and o Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* 16.42 (2015), pp. 1437–1480.

[18] Sven Gronauer and Klaus Diepold. "Multi-agent deep reinforcement learning: a survey". In: *Artificial Intelligence Review* (2021). DOI: 10.1007/s10462-021-09996-w.

[19] Sven Gronauer, Martin Gottwald, and Klaus Diepold. "The Successful Ingredients of Policy Gradient Algorithms". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Main Track. Aug. 2021, pp. 2455–2461. DOI: 10.24963/ijcai.2021/338.

[20] Sven Gronauer et al. "Using Simulation Optimization to Improve Zero-shot Policy Transfer of Quadrotors". In: *ArXiv* abs/2201.01369 (2022).

[21] Peter Henderson et al. *Deep Reinforcement Learning That Matters*. 2018.

[22] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[23] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning." In: *ICLR (Poster)*. 2016.

[24] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (Feb. 2015), 529 EP -.

[25] Teodor Mihai Moldovan and Pieter Abbeel. "Safe Exploration in Markov Decision Processes". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 1451–1458.

[26] Martin Pecka and Tomas Svoboda. "Safe Exploration Techniques for Reinforcement Learning – An Overview". In: *Modelling and Simulation for Autonomous Systems*. Ed. by Jan Hodicky. Cham: Springer International Publishing, 2014, pp. 357–375.

[27] X. B. Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[28] Alex Ray, Joshua Achiam, and Dario Amodei. "Benchmarking Safe Exploration in Deep Reinforcement Learning". In: (2019).

[29] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *CoRR* abs/1506.02438 (2015).

[30] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017).

[31] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1889–1897.

[32] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (Jan. 2016), 484 EP -.

[33] Adam Stooke, Joshua Achiam, and Pieter Abbeel. “Responsive Safety in Reinforcement Learning by PID Lagrangian Methods”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 9133–9143.

[34] Yuval Tassa et al. “DeepMind Control Suite”. In: *CoRR* abs/1801.00690 (2018).

[35] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. “Reward Constrained Policy Optimization”. In: *International Conference on Learning Representations*. 2019.

[36] J. Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[37] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[38] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[39] Tsung-Yen Yang et al. “Projection-Based Constrained Policy Optimization”. In: *International Conference on Learning Representations*. 2020.

[40] Yiming Zhang, Quan Vuong, and Keith Ross. “First Order Constrained Optimization in Policy Space”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 15338–15349.

Appendix

In this supplementary material, we elaborate on the experimental approach and explain the hyper-parameter configuration of our experiments in detail. We depict an overview of the applied hyper-parameters in Table 2. Further, we include the learning curves of all evaluated hyper-parameter configurations separately for each algorithm (see Fig. 3-7).

Experimental Setup and Hyper-parameters

For all experiments, we used the hyper-parameters listed in Table 2 as default. The network structure for both policy and value network was a multi-layer perceptron with two hidden layers each consisting of 64 neurons and followed by tanh non-linearities. Weights were not shared between value and policy network. This configuration followed the one suggested in [19].

We considered an on-policy settings where the agent interacted sequentially B-times with the environment and then performed a policy update step based on this experience batch before generating a new experience batch of size B. This modus operandi was repeated for M epochs after which the training ended and the performance of the agent was evaluated. Over the training, actions were sampled from a Gaussian distribution $\mathcal{N}(\pi(x), \varepsilon I)$, where the mean is given by the policy π and εI is the co-variance matrix defined by the scalar ε and the identity I. During the training, we annealed ε towards zero to promote the fulfillment of safety constraints of a deterministic policy at the end of training. For evaluation purposes, we deactivated the exploration noise so that the stochastic policy function became a deterministic function of the state, i.e. $u = \pi(x)$ instead of $u \sim \mathcal{N}(\pi(x), \varepsilon I)$.

For all experiments, we employed a distributed learner setup where policy gradients are computed and averaged across all distributed processes. We used Threadripper 3990X CPUs, which are capable to run 64 distributed MPI processes in parallel. All algorithms were run on a *Ubuntu 20.04.2 LTS* operating system.

In order to render the evaluation process comparable, we searched over a grid of hyper-parameters for each algorithm and determined the performance based on the average over four independent random seeds. We searched over the following settings:

1. *TRPO:* $\delta \in \{0.001, 0.01\}$
2. *TRPO-L:* $\delta \in \{0.0001, 0.001, 0.01\}$ and $\alpha_\lambda \in \{0.001, 0.01, 0.1\}$
3. *CPO:* $\delta \in \{0.0001, 0.0005, 0.001\}$ and $\lambda_c \in \{0.5, 0.9, 0.95\}$
4. *PDO:* $\delta \in \{0.0001, 0.001, 0.01\}$ and $\alpha_\lambda \in \{0.001, 0.01, 0.1\}$
5. *PCPO:* $\delta \in \{0.0001, 0.0005, 0.001\}$ and $\lambda_c \in \{0.5, 0.9, 0.95\}$

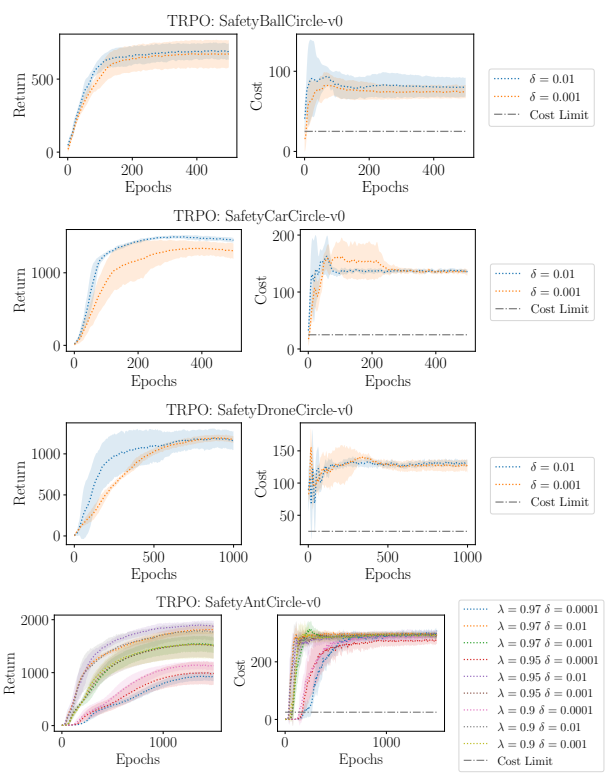
Note that the Lagrange multiplier was optimized with SGD for the TRPO-L algorithm. We observed that Adam generated stronger cost oscillation whereas SGD was more stable. In contrast, the PDO algorithm took profit from Adam in terms of the cost violations.

Table 2: Default hyper-parameters used in all experiments. The values marked with * were determined through grid search while values marked with # depend on environment specifics (see Sect. 6).

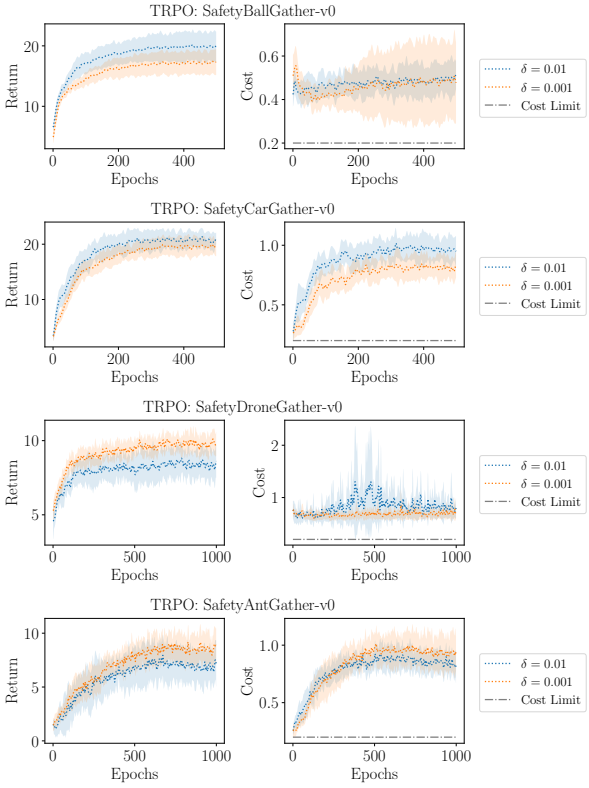
Hyper-parameter	TRPO	TRPO-L	CPO	PDO	PCPO
Backtracking budget	15	15	25	N/A	25
Backtracking decay	0.8	0.8	0.8	N/A	0.8
Batch-size B	#	#	#	#	#
Conjugate grad. damping	0.1	0.1	0.1	0.1	0.1
Conjugate grad. iterations	10	10	10	10	10
Cost limit d	#	#	#	#	#
Discount factor γ	0.99	0.99	0.99	0.99	0.99
Entropy co-efficient	0	0	0	0	0
Exploration noise ε (init.)	0.5	0.5	0.5	0.5	0.5
GAE factor costs λ_c	0.95	0.95	*	0.95	*
GAE factor rewards λ	0.95	0.95	0.95	0.95	0.95
Lagrangian learn rate α_λ	N/A	*	N/A	*	N/A
Lagrangian optimizer	N/A	SGD	N/A	Adam	N/A
Target KL divergence δ	*	*	*	*	*
Training Epochs N	#	#	#	#	#
V-network mini-batch size	64	64	64	64	64
V-network updates	80	80	80	80	80
V-network learning rate	0.001	0.001	0.001	0.001	0.001
V-network optimizer	Adam	Adam	Adam	Adam	Adam
Weight initialization gain κ	$\sqrt{5}$	$\sqrt{5}$	$\sqrt{5}$	$\sqrt{5}$	$\sqrt{5}$
Weight initialization	Kaim.	Kaim.	Kaim.	Kaim.	Kaim.

Table 3: Overview of proposed environments along with their specification.

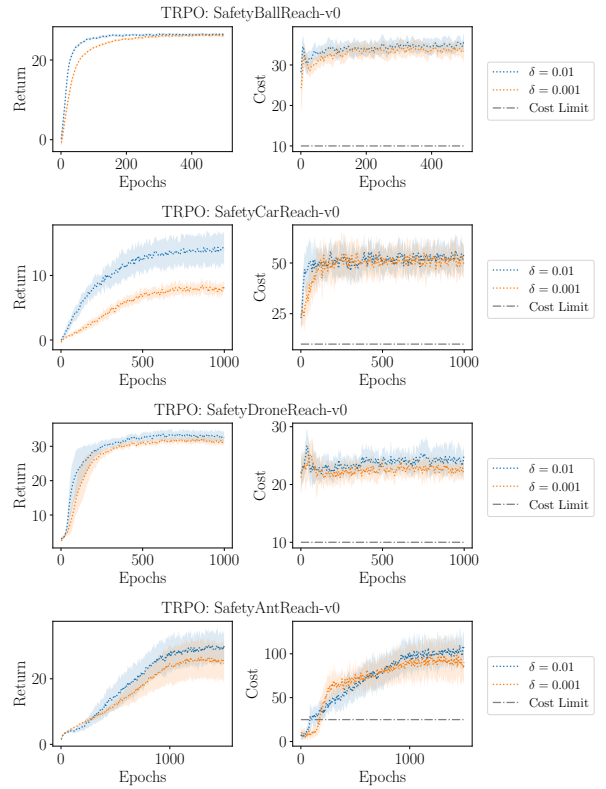
Environment Name	Cost limit d	Epochs M	Batch Size B	Maximum Episode Length
SafetyBallRun-v0	25	100	32000	250
SafetyCarRun-v0	25	200	32000	500
SafetyDroneRun-v0	25	500	64000	500
SafetyAntRun-v0	25	500	64000	1000
SafetyBallCircle-v0	25	500	32000	250
SafetyCarCircle-v0	25	500	32000	500
SafetyDroneCircle-v0	25	1000	64000	500
SafetyAntCircle-v0	25	1000	64000	1000
SafetyBallGather-v0	0.2	500	32000	250
SafetyCarGather-v0	0.2	500	32000	500
SafetyDroneGather-v0	0.2	1000	64000	500
SafetyAntGather-v0	0.2	1000	64000	1000
SafetyBallReach-v0	10	500	32000	250
SafetyCarReach-v0	10	1000	32000	500
SafetyDroneReach-v0	10	1000	64000	500
SafetyAntReach-v0	25	1500	64000	1000



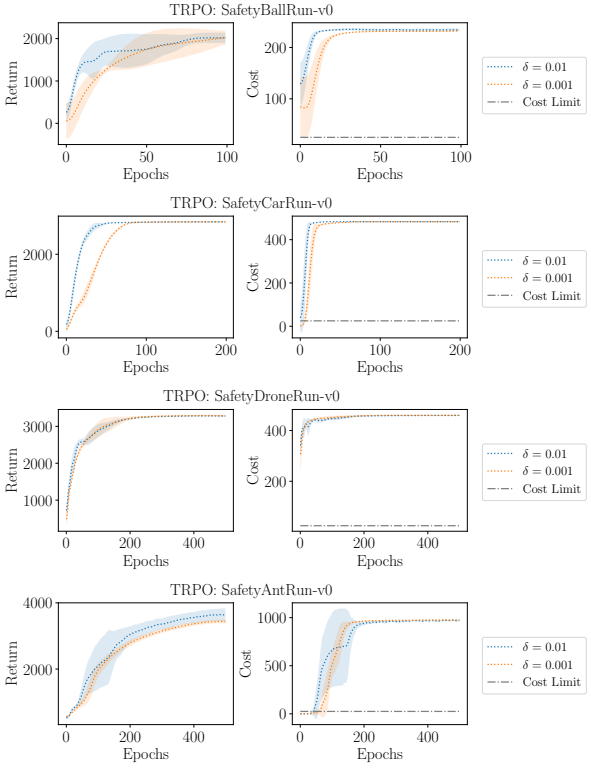
(a) Circle tasks.



(b) Gather tasks.

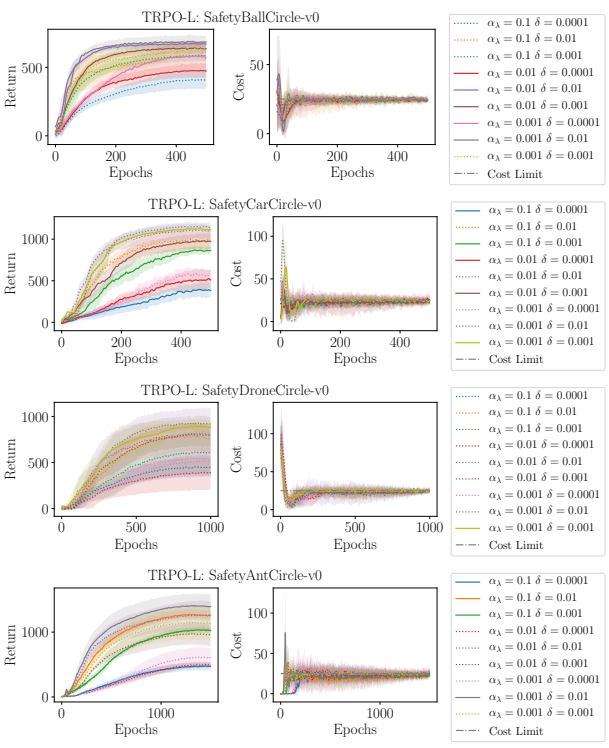


(c) Reach tasks.

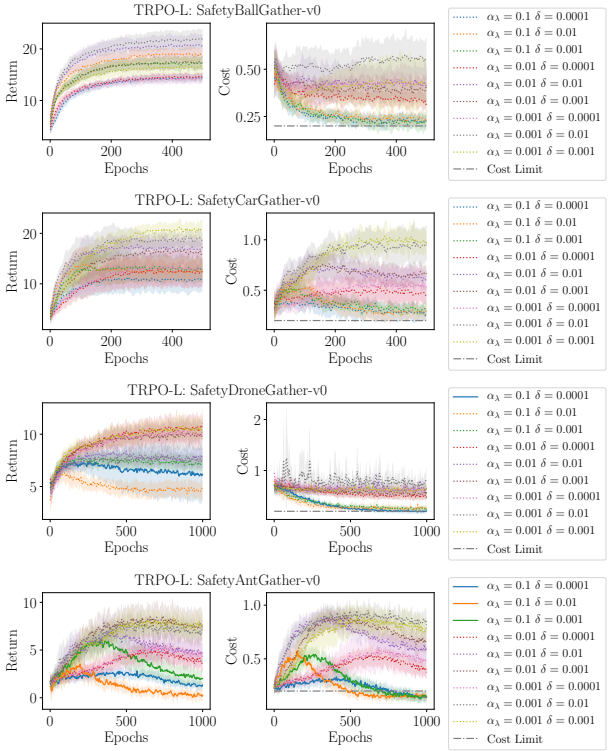


(d) Run tasks.

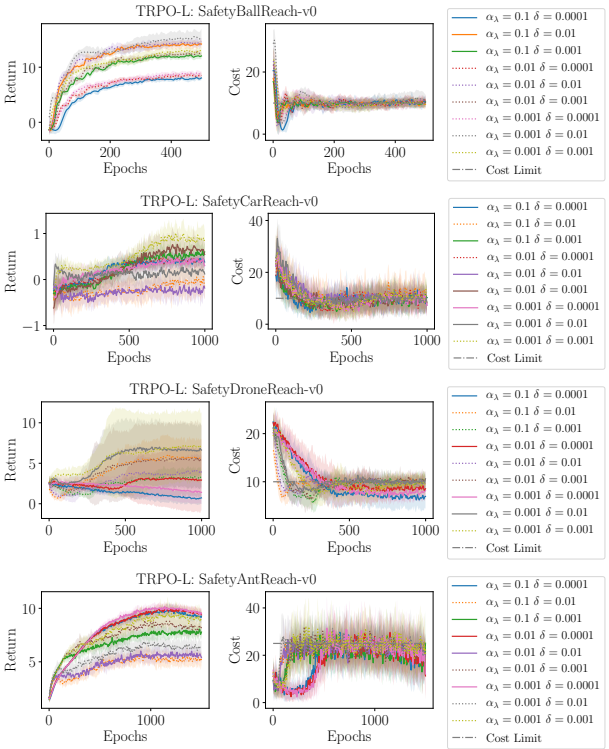
Figure 3: TRPO. Each hyper-parameter configuration was averaged four independent random seeds. Thick lines indicate that the configuration was able to fulfill the cost constraint on average while dotted lines denote cost constraint violation.



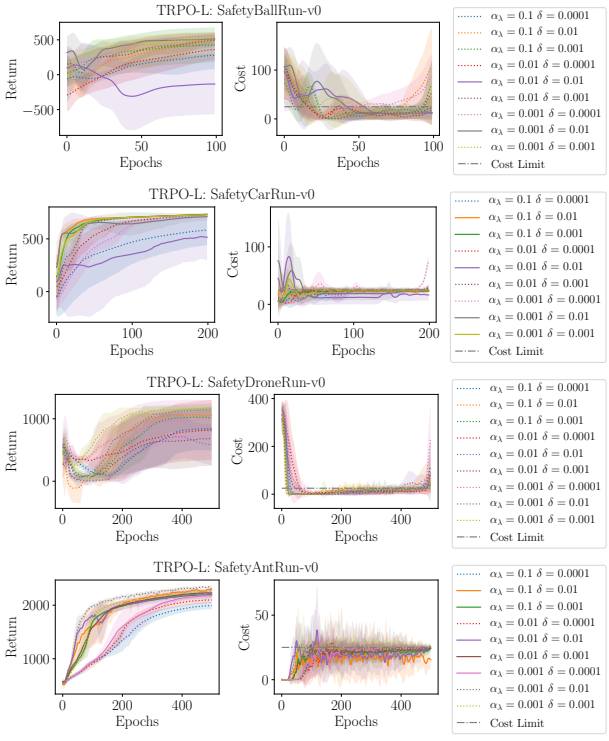
(a) Circle tasks.



(b) Gather tasks.

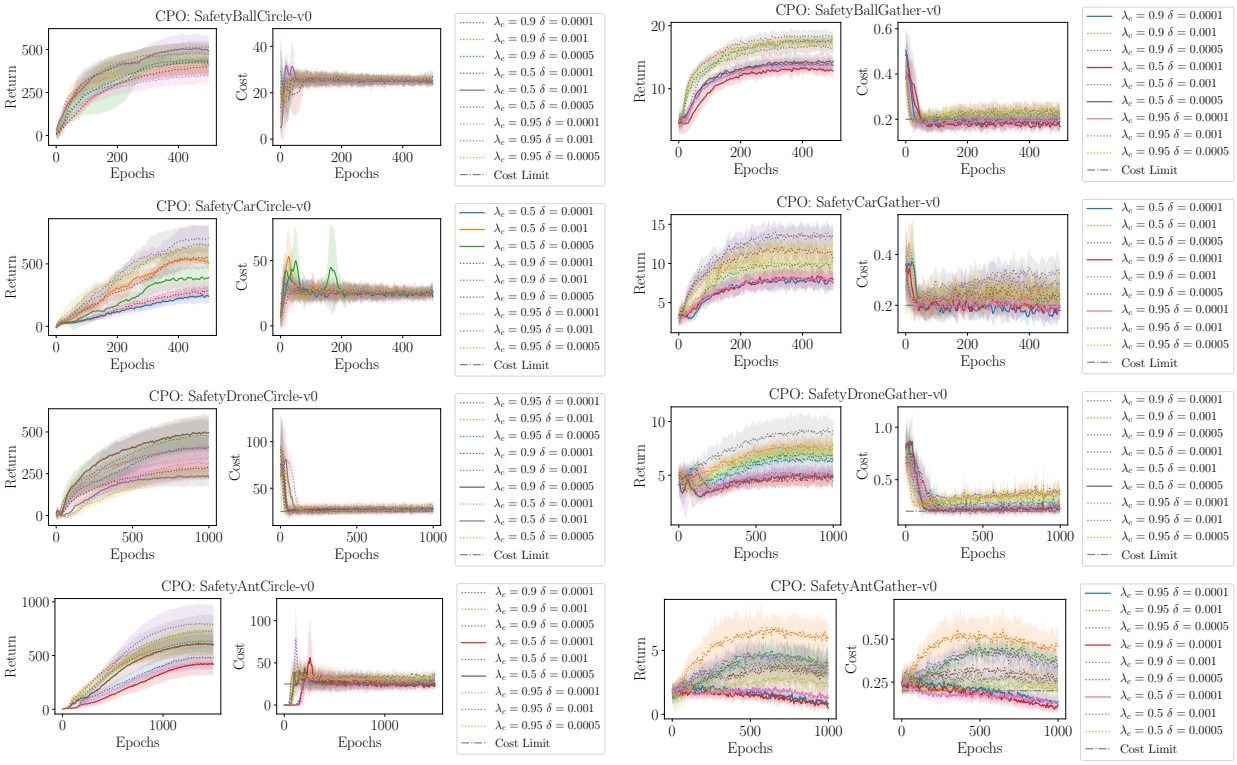


(c) Reach tasks.



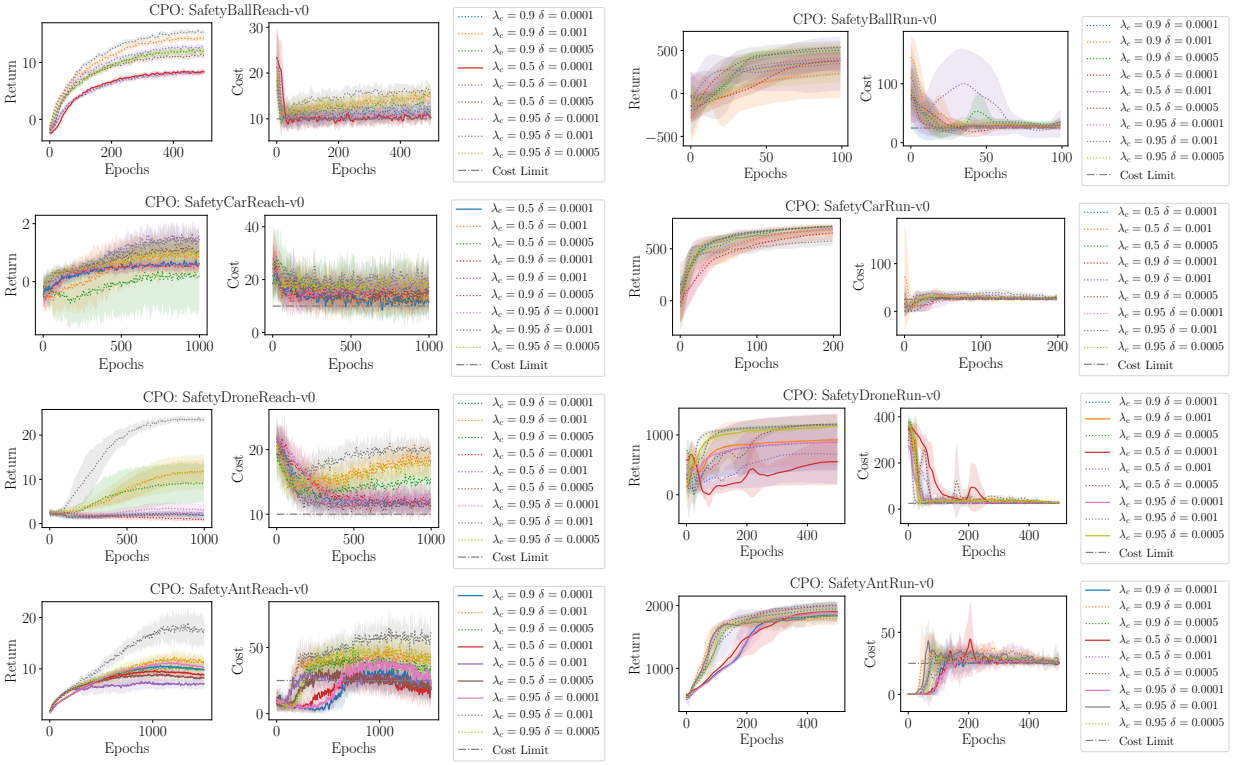
(d) Run tasks.

Figure 4: TRPO-L. Each hyper-parameter configuration was averaged four independent random seeds. Thick lines indicate that the configuration was able to fulfill the cost constraint on average while dotted lines denote cost constraint violation.



(a) Circle tasks.

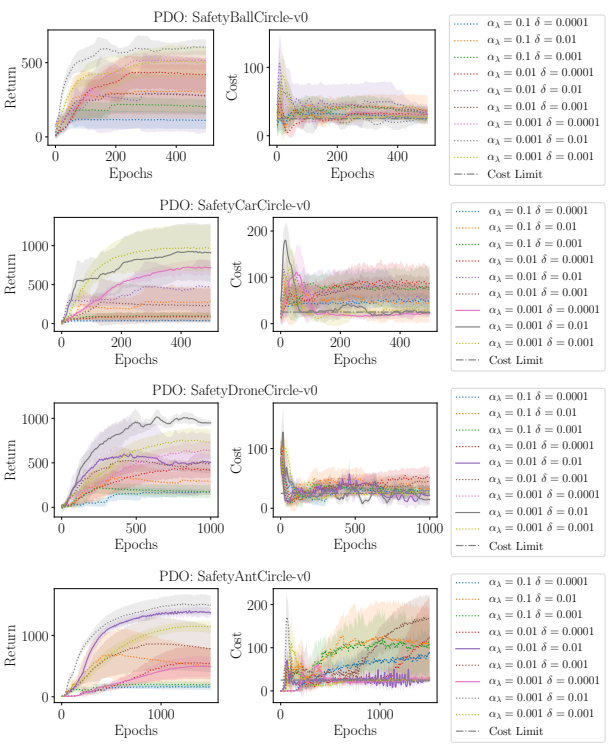
(b) Gather tasks.



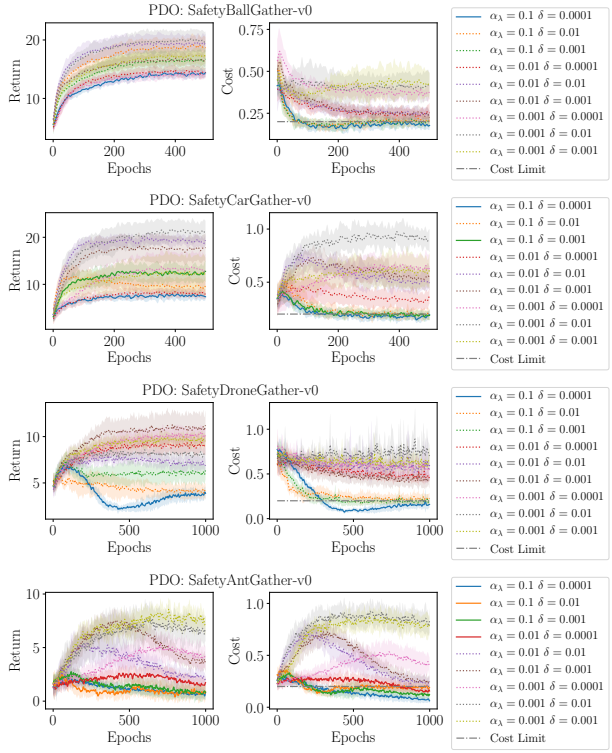
(c) Reach tasks.

(d) Run tasks.

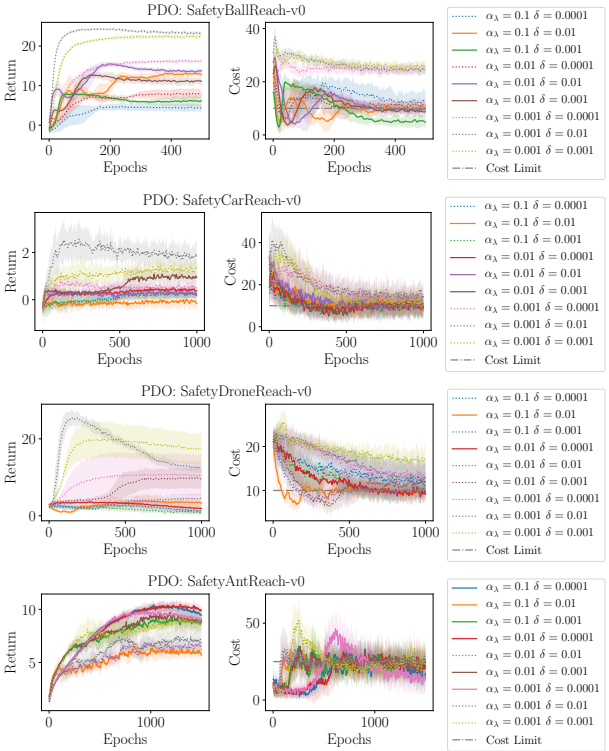
Figure 5: CPO. Each hyper-parameter configuration was averaged four independent random seeds. Thick lines indicate that the configuration was able to fulfill the cost constraint on average while dotted lines denote cost constraint violation.



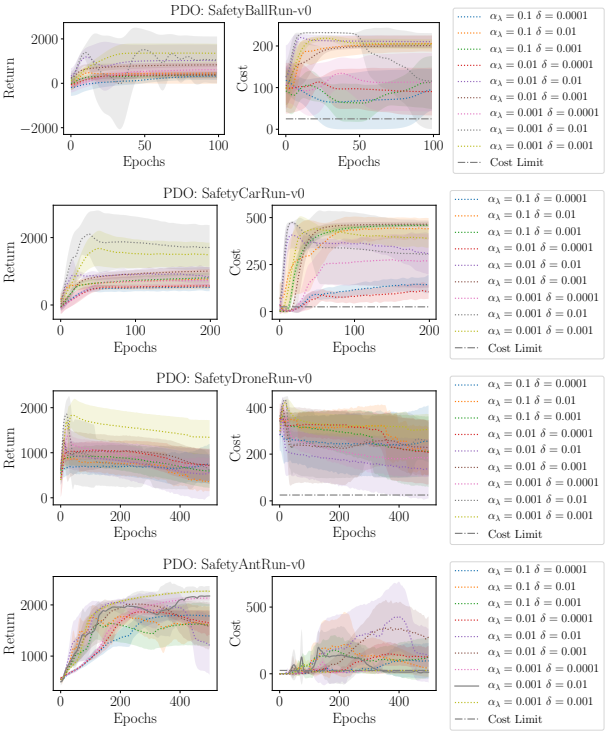
(a) Circle tasks.



(b) Gather tasks.

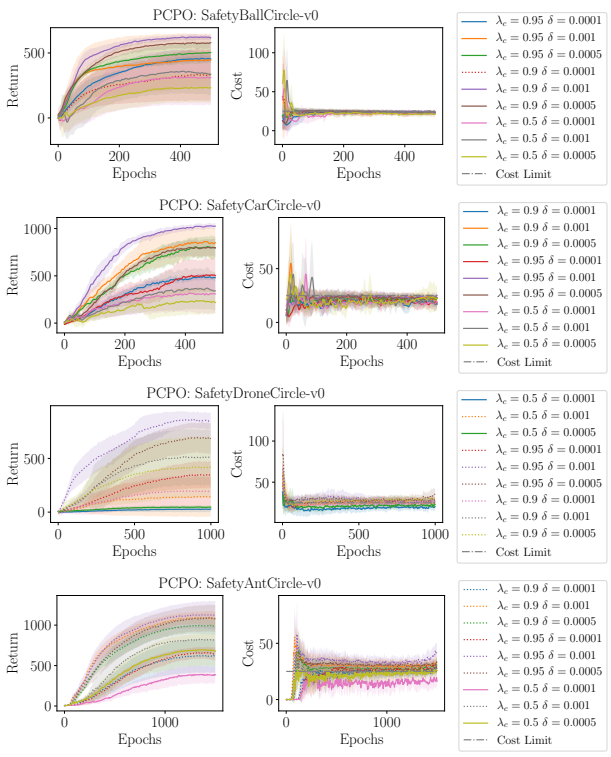


(c) Reach tasks.

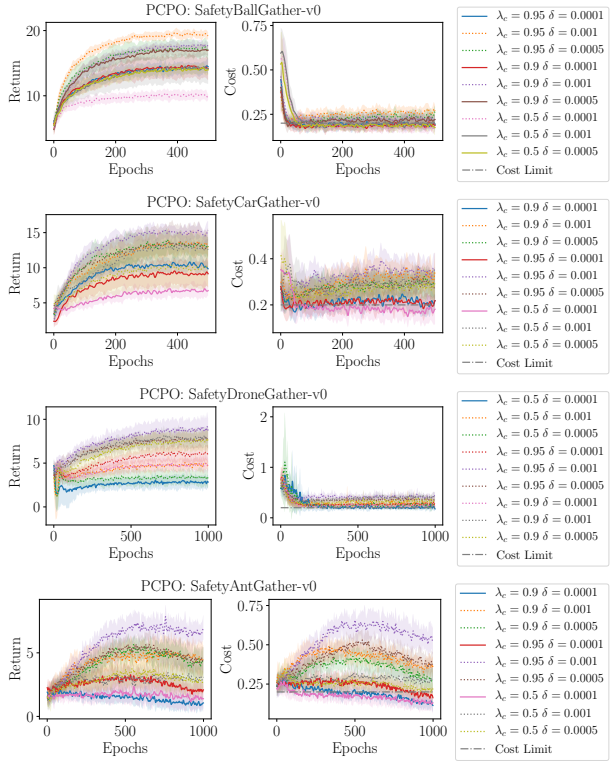


(d) Run tasks.

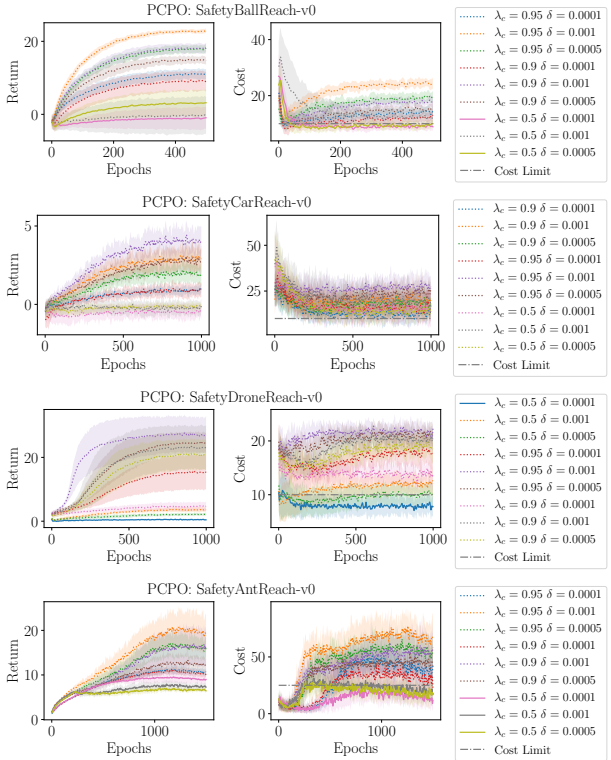
Figure 6: PDO. Each hyper-parameter configuration was averaged four independent random seeds. Thick lines indicate that the configuration was able to fulfill the cost constraint on average while dotted lines denote cost constraint violation.



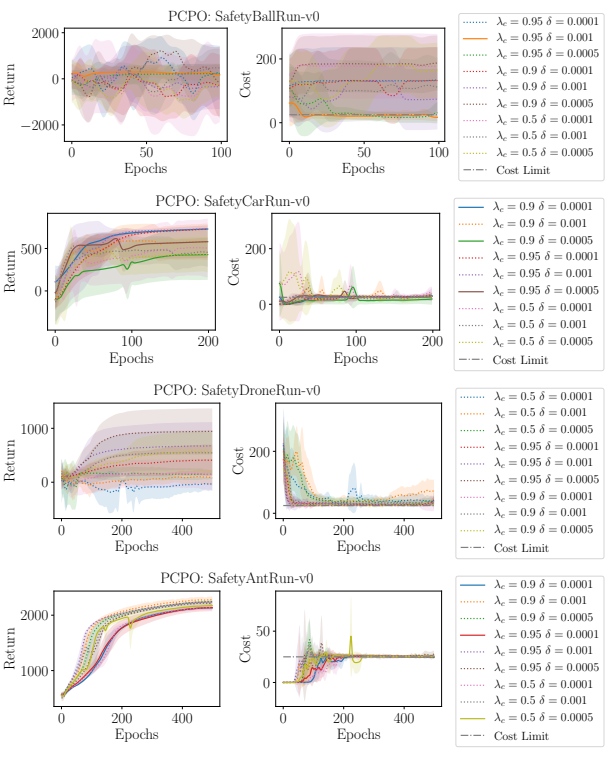
(a) Circle tasks.



(b) Gather tasks.



(c) Reach tasks.



(d) Run tasks.

Figure 7: PCPO. Each hyper-parameter configuration was averaged four independent random seeds. Thick lines indicate that the configuration was able to fulfill the cost constraint on average while dotted lines denote cost constraint violation.