Exploring Safe Reinforcement Learning for Sequential Decision Making

Fan Yang

CMU-RI-TR-23-22

May 18, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
David Held, *chair*
Guanya Shi

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Keywords: Robotics, Reinforcement Learning, Safe Reinforcement Learning, Safety, Machine Learning, Curriculum, Trajectory Optimization, Constraint

To my family.

Abstract

Safe Reinforcement Learning (RL) focuses on the problem of training a policy to maximize the reward while ensuring safety. It is an important step towards applying RL to safety-critical real-world applications. However, safe RL is challenging due to the trade-off between the two objectives of maximizing the reward and satisfying the safety constraints, which could lead to unstable training and over-conservative behaviors.

In this thesis, we propose two methods of solving the issues mentioned above in safe RL:

(1) We propose Self-paced Safe Reinforcement Learning which combines a self-paced curriculum on the safety objective with a base safe RL algorithm PPO-Lagrangian. During training, the policy starts with easy safety constraints and gradually increases the difficulty of the constraints until the desired constraints are satisfied. We evaluate our algorithm on the Safety Gym benchmark and demonstrate that the curriculum helps the underlying Safe RL algorithm to avoid local optima and improves the performance for both reward and safety objectives.

(2) We propose to learn a policy in a modified MDP in which the safety constraints are embedded into the action space. In this "safety-embedded MDP," the output of the RL agent is transformed into a sequence of actions using a trajectory optimizer that is *guaranteed* to be safe, under assumption that the robot is currently in a safe and quasi-static configuration. We evaluate our method in the Safety Gym benchmark and show that we achieve significantly higher rewards and fewer safety violations during training than previous work; further, we have no safety violations during inference. We also evaluate our method on a real robot box-pushing task and demonstrate that our method can be safely deployed in the real world.

Acknowledgments

I would like to thank my family, especially my parents who support me emotionally and financially for my study at CMU. I am never able to come back home for two years due to visa restrictions and workload. My parents are gradually getting old and they cannot not enjoy the pleasure of having their kid around. My parents cut down their daily expenses to support my study in the US as much as possible. I was not able to provide any help when they were facing great stress emotionally and financially. I appreciate a lot for their scarification and will try my best to give them a happier life in the future.

I would also like to thank my friends at CMU who help me through my darkest time. If I had never met you, I would become so depressed and have nothing to do other than working on the research project I am having. It was you that I felt the hope and pleasure of my life at Pittsburgh. And I have learned a lot from you about how to communicate with people and how to keep idealistic and romantic when we pursue our life goals.

I would also like to thank my advisor Dave, who introduces me to safe RL. It was an enriching experience researching in safe RL. Without his help, I will never make such a progress.

Finally, thank United States United States Air Force and DARPA, National Science Foundation and CMU Argo AI Center for Autonomous Vehicle Research for supporting my research financially.

Funding

This material is based upon work supported by the United States United States Air Force and DARPA under Contract No. FA8750-18-C-0092.

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1849154.

This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations and Challenges

Reinforcement learning (RL) has shown remarkable progress in sequential decision making tasks in the past few years [2, 38, 51], demonstrating the potential of achieving human-level dexterity in robotic tasks. To apply RL into robotics tasks in the real world, in addition to successfully achieving the desired task, it is also critical to ensure that the robot satisfies the safety constraints to avoid damage to the robot and humans.

Although there has been a significant emphasis on achieving more dexterous behaviors for robots using RL recently, ensuring safety guarantee still remains one of the bottlenecks for applying RL into real world, regardless of the achievements in making robots more intelligent and dexterous. However, how to ensure safety during deployment for reinforcement learning policy is still an ongoing research direction. While training RL policy to maximize the reward is a ch alleging task itself, including constraints over RL optimization creates an additional layer of difficulty for optimization. Specifically, the RL policy optimization needs to leverage between reward maximization and constraints satisfaction, which could lead to unstable training and local optima convergence.

Several attempts have been made to solve policy optimization under safety constraints. The problem is generally formulated as a Constrained Markov Decision Process. Lagrangian methods [46, 57] has been a major category for solving CMDP,

but those methods usually struggle with training stability and can generally not guarantee the desired safety requirements. Another line of work in safe RL will incorporate a barrier function or shielding function [3, 57], which is designed to correct the unsafe actions to guarantee safety. However, those methods usually requires additional information such as dynamics model. There are also methods in the domain of trajectory optimization and motion planning [24, 25], however, those methods could struggle with planning horizons, and may need additional computation to achieve safe performance on long-horizon tasks.

In this thesis, we propose two methods of improving safety in RL. We first design "self-paced safe reinforcement learning", which incorporates curriculum learning into the RL training to mitigate the optimization challenge of training a safe RL agent. Specifically, the algorithm starts from a much "softer" constraints to encourage exploration for RL algorithm. The constraints are gradually tightened until the desired constraints are achieved.

We also propose "Safety-Embedded Markov Decision Process", in which a trajectory optimization module is embedded into a MDP to ensure safety, the higher level RL algorithm outputs a subgoal and it only needs to maximize the reward. Safety-Embedded MDP combines the benefits from both RL and control, enabling us to improve safety and deal with complex interactions with the environment.

1.2 Thesis Organization

We organize the thesis as follows: In Chapter 2, we present "Self-Paced Safe Reinforcement Learning". The chapter is divided into the following sections: (1) In the introduction section, we discuss the challenge of safe RL, the recent progress in curriculum learning and how we could combine them together to improve the performance of safe RL. (2) Related works in safe RL and curriculum learning. (3) We discuss the preliminaries, including MDP, Constrained MDP and the PPO Lagrangian method. (4) We present the framework of our method (5) The experiments in Safety Gym environment. More experiments can be found in Chapter A

In Chapter 3, we present "Safety-Embedded Markov Decision Process". (1) In the introduction section, we discuss the challenges in safe RL and the potential of combining RL with trajectory optimization. (2) We discuss the related works. (3) We

present the framework of our method and the details in trajectory optimizer design. (4)We evaluate our method and baseline methods in Safety Gym environments and a real-world robot setup. More experiments are shown in Chapter B.

Chapter 2

Self-Paced Policy Optimization with Safety Constraints

2.1 Introduction

Reinforcement Learning has demonstrated a lot of success in sequential decision making tasks [38, 51]. In most cases, RL algorithms only have a single objective of maximizing the reward during training. However, when safety is a concern in some domains such as human-robot interaction [36, 41] or autonomous driving [26, 39, 53], the safety objective is incorporated into the RL objective as a constraint. Thus, the policy will be optimized to maximize the reward while satisfying the constraints.

However, the trade-off between the reward objective and the safety constraints will create difficulties for training [46]. For example, the policy tends to get stuck at an over-conservative local optima where the safety constraints are satisfied with low rewards. For example, in autonomous driving tasks, the agent is likely not to move at all to avoid any collision with the obstacles.

In this work, we aim to improve the suboptimal solutions in safe RL with curriculum learning. Curriculum learning [10, 16] has demonstrated the abilities of stabilizing the training, increasing sample efficiency and avoiding local optima. Previous work [22] has demonstrated that curriculum learning can effectively modify the optimization landscape and converge to better final performance. These properties can

help the optimization difficulties in safe RL. In addition, to avoid manually designing the curriculum, we consider automatic curriculum [2] or self-paced learning [32], where the algorithm automatically adjusts the curriculum based on its current performance.

More specifically, we build on top of Lagrangian penalized versions of PPO and propose to improve its performance by applying self-paced learning on the safety constraints. The agent is trained with relaxed constraints first and chooses more challenging constraints when the current constraints are easy to satisfy. For example, the agent can first learn to drive fast and then learn to drive safely to avoid the local optima of driving safely but slowly. This type of training procedure can help exploration and converge to a better final performance in a lot of scenarios. We emphasize that our objective in this work is to obtain a high performance safe policy at deployment but not during training. For example, with sim2real transfer, we can learn a policy aggressively in simulation and deploy a safe policy on the real robot.

In summary, we propose a self-paced policy optimization algorithm with safety constraints which could improve exploration and lead to better final performances. We demonstrate the effectiveness of our algorithm on a toy example on constraint optimization and evaluate it on the Safety Gym benchmark [46].

2.2 Related Work

2.2.1 Safe RL

Safety is an important consideration in a lot of real-world applications. The goal of safe RL is to train RL policies with safety constraints in addition to the reward objective. Lagrangian method has been widely used in solving Safe RL problems [4, 13, 19]. The Lagrangian method defines a Lagrangian function which combines reward and cost functions and automatically adjusts the weights between the two objectives. Constrained Policy Optimization (CPO) [1] derives the trust region for the update in safe RL setting and can guarantee monotonic policy improvement while satisfying safety constraints. Other methods apply Lyapunov constraints to ensure a safe policy update [11, 14, 15, 42, 50]. However, existing safe RL algorithms still struggle with the trade-off between reward and safety constraints. In addition, methods such as CPO also aim to ensure safety during the exploration process which could be over-

conservative. Our method is built on top of PPO-Lagrangian method and improves the exploration with self-paced safety constraints.

2.2.2 Curriculum Learning

Curriculum learning [16] has been shown to be beneficial in both supervised learning and RL [10, 58]. In addition, generating a curriculum automatically can lead to better asymptotic performance compared to a fixed curriculum. For example, Kumar et al. [32] propose the idea of self-paced learning. In their method, the easiness of each task is defined by computing the loss of it. The training will focus more on tasks that could be easily achieved for the current network. Florensa et al. [18] and Graves et al. [20] also propose the idea of automatically generating the curriculum. Another recent method, Automatic Domain Randomization [2] allows the agent to increase the difficulties of the tasks automatically based on its current policy.

2.3 Background

2.3.1 Markov Decision Process

A Markov decision process (MDP) is described as a tuple $(S, A, r, P, \rho, \gamma)$, where S is the state set, A is the action set, $r(\mathbf{s}, \mathbf{a}) : S \times A \longrightarrow \mathbb{R}$ is the reward function. $P : S \times A \times S \longrightarrow [0, 1]$ is the state transition probability function, ρ denotes the initial state distribution, and γ is the discount factor. Specifically, each episode starts with an initial state $\mathbf{s}_0 \sim \rho(\mathbf{s}_0)$, the state is input into a parameterized policy π_θ to get the action $\mathbf{a}_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$ The agent takes the action $\mathbf{a}_t$ and the next state is sampled from the environment $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. For a given state-action tuple, the reward is given as $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$. The training objective is to maximize the expected discounted sum of reward:

$$\max_\theta J_r(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)], \tag{2.1}$$

where $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ...)$ is the trajectory sampled given the policy π_θ, initial state distribution $\mathbf{s}_0 \sim \rho(\mathbf{s}_t)$, and the state transition function $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$.

2.3.2 Constrained Markov Decision Process

A constrained Markov decision process (CMDP) is an augmented version of MDP [5]. Specifically, a constraint function $c(\mathbf{s}_t, \mathbf{a}_t)$ are added to MDP, where $c(\mathbf{s}_t, \mathbf{a}_t) : S \times A \longrightarrow \mathbb{R}$. Without loss of generality, we assume that constraint is defined as $c(x) \leq 0$(Any $c(x) \leq d$ could be rewritten as $c'(x) = c(x) - d \leq 0$.). Similarly, we could also define the expected discounted cost as:

$$J_c(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{\infty} \gamma^t c(\mathbf{s}_t, \mathbf{a}_t)]. \tag{2.2}$$

Then the training objective is defined as:

$$\max_{\pi_\theta \in \Pi_f} J_r(\pi_\theta). \tag{2.3}$$

where $\Pi_f \doteq \{\pi_\theta \in \Pi : J_c(\pi_\theta) \leq 0\}$ is the feasible set.

2.3.3 PPO-Lagrangian Method

Lagrangian method [4] is one of widely used methods for solving constrained optimization problems. Specifically, in our problem, we define the Lagrangian function $L(\theta, \boldsymbol{\lambda})$ as:

$$L(\theta, \lambda) = -J_r(\pi_\theta) + \lambda \cdot J_c(\pi_\theta) \tag{2.4}$$

The training objective can be written as:

$$\max_{\lambda \geq 0} \min_{\theta} L(\theta, \lambda) \tag{2.5}$$

PPO-Lagrangian combines the Lagrangian method with Proximal Policy Optimization (PPO) [49]. PPO defines the training objective as:

$$\begin{aligned} L_r^{\theta_k}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_k}}[\min(&\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_k}(\mathbf{a}|\mathbf{s})} A_r^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}), \\ &\text{clip}(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_k}(\mathbf{a}|\mathbf{s})}, 1-\epsilon, 1+\epsilon) A_r^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}))], \end{aligned} \tag{2.6}$$

Algorithm 1 Self-Paced PPO-Lagrangian Method

Input: number of training epochs N_t, number of policy training epochs N_π.
for $n = 1$ **to** N_t **do**
  Estimate the expected discounted sum of cost $J_c(\pi_\theta)$
  $d_i \longleftarrow \text{argmax}_{d_i \in D}\, d_i \leq J_c(\pi_\theta)$
  $\lambda \longleftarrow \lambda + \beta * (J_c(\pi_\theta) - d_i)$
  for $t = 1$ **to** N_π **do**
    $\theta \longleftarrow \theta + \alpha * \nabla_\theta(-L_r^{\theta_k}(\theta) + \lambda \cdot L_c^{\theta_k}(\theta))$
  end for
end for

where $A_r^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a})$ is the advantage function. PPO-Lagrangian uses a similar formulation for reward to define the training objectives $L_c^{\theta_k}(\theta)$ for cost. Then the training objective of PPO-Lagrangian is defined as:

$$\max_{\lambda \geq 0} \min_{\theta} -L_r^{\theta_k}(\theta) + \lambda \cdot L_c^{\theta_k}(\theta) \tag{2.7}$$

2.4 Self-Paced Safe Reinforcement Learning (SPSRL)

2.4.1 SPSRL Framework

As discussed in Sec 2.2, directly optimizing Eq. 2.3 is challenging because it suffers from local optima and training instabilities. We propose self-paced policy optimization with safety constraints, which enables the agent to choose the suitable training objectives automatically and alleviate the problems mentioned above.

Ideally, when an agent is presented tasks with different difficulties, the agent should gradually learn from the easy ones to the hard ones. In our case, we define a task in safe RL with different difficulties by setting different safety threshold d_i: $D = \{d_1, d_2, ..., d_k\}$. We rank the thresholds with d_1 as the lowest one and d_k as the highest one. During deployment, we may only care about one of the threshold defined in D, which is usually the smallest one $d_1 = 0$. The other thresholds are defined to help the training procedure.

During training, we start from the highest threshold d_k. As long as the safety

threshold is satisfied: $J_c(\pi_\theta) < d_k$, we decrease the threshold to d_{k-1}. We iterate such procedure until the minimal threshold d_0 is satisfied. Formally, we solve the following equations iteratively:

$$\min_\theta -J_r(\pi_\theta) \quad \text{s.t.} \quad J_c(\pi_\theta) \leq d_i, \tag{2.8}$$

where

$$d_i = \underset{d_i \in D}{\text{argmax}} \; d_i \leq J_c(\pi_\theta). \tag{2.9}$$

2.4.2 Self-Paced PPO-Lagrangian method

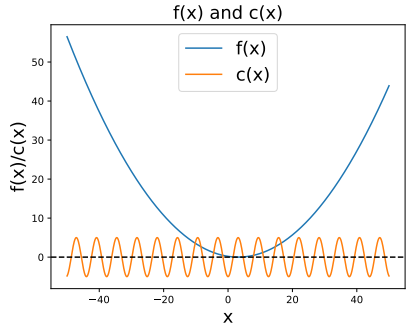
In this work, we combine the proposed self-paced curriculum with PPO-Lagrangian. However, a similar idea could potentially be applied to other constrained optimization algorithms, and we leave that exploration to future work. To solve the max-min optimization in PPO-Lagrangian, we have two iterations for parameter update: one updates policy parameters θ with higher frequency and the other updates λ with much lower frequency. Note that different threshold will only affect the maximization over λ but will not affect the minimization over θ. This is because adding a constant value will not affect the gradient update of θ. Thus, at each iteration, λ is updated according to:

$$\lambda_{t+1} = \lambda_t + \beta * (J_c(\pi_\theta) - d_i), \tag{2.10}$$

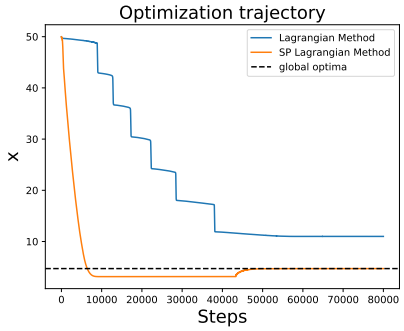
where d_i is defined as Eq. 2.9, and β is the update rate for λ. The algorithm is summarized in Alg. 1.

2.5 Experiments

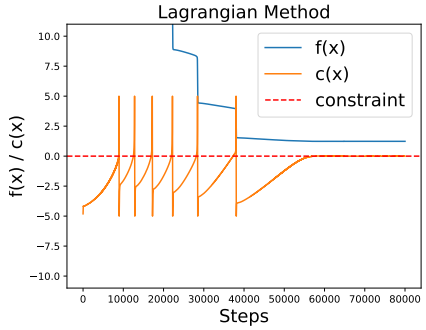
In this section, we first study a toy example to demonstrate the benefits of using self-paced learning on constraints. Then we evaluate the combination of self-paced constraints with PPO-Lagrangian on the Safety Gym Benchmark [46]. More results can be found in Appendix A.1.



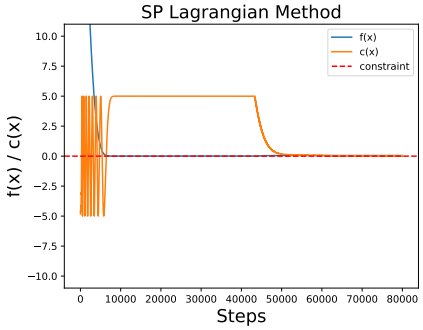
(a) $f(x)$ and $c(x)$



(b) Optimization Trajectory

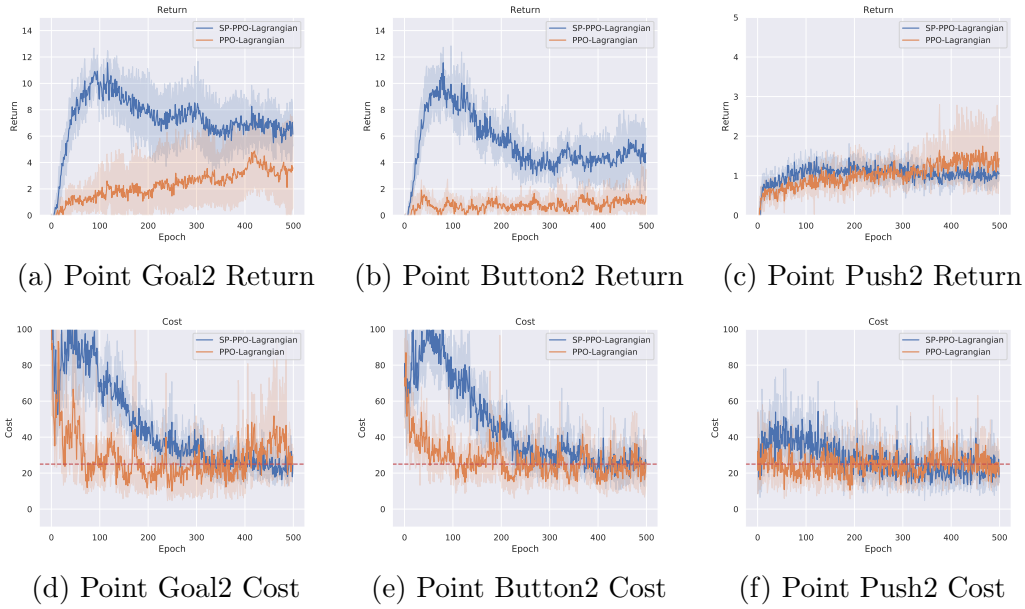


(c) Lagrangian Method



(d) SP Lagrangian Method

Figure 2.1: Fig. 2.1a visualizes $f(x)$ and $c(x)$. Fig. 2.1b visualizes the trajectory of x during optimization. Fig. 2.1c and Fig. 2.1d visualizes the trajectory of $f(x)$ and $c(x)$. The Lagrangian method converges to a local optima which satisfies the constraint but does not achieve the best possible performance. The self-paced Lagrangian method would relax the constraint first and finally converge to a global optima which is both feasible and optimal.



(a) Point Goal2 Return (b) Point Button2 Return (c) Point Push2 Return

(d) Point Goal2 Cost (e) Point Button2 Cost (f) Point Push2 Cost

Figure 2.2: PPO-Lagrangian (orange curve) and SP-PPO-Lagrangian (blue curve) are evaluated in the Safety Gym environment. We choose the point agent and different levels of goal, button, and push tasks. In most of the tasks, our method can converge to a policy with better final returns while has a similar performance as PPO-Lagrangian in terms of satisfying the safety constraints.

2.5.1 Toy Example

We first apply self-paced learning with safety constraints on a toy example to give an intuition why self-paced learning would help constrained optimization. In this experiment, we formulate our problem as the following objective:

$$\min_x f(x), \quad \text{s.t. } c(x) \leq 0. \tag{2.11}$$

Ideally, the proposed self-paced learning over the constraints is supposed to be effective when the feasible set is composed of multiple disconnected subsets. In this case the algorithm may be trapped into a certain constraint set and cannot find the global optima. Thus, we define the optimization objective function $f(x) = \frac{1}{50}(x - \pi)^2$ and the constraint function $c(x) = -5\cos(x)$. Mathematically, the global optima should be $x = \frac{\pi}{2}$ or $x = \frac{3\pi}{2}$.

We first apply Lagrangian method to solve Eq. 2.11. Then we combine SPCPO with Lagrangian method (denoted as SP Lagrangian method). The threshold set D is defined as:

$$D = \{d_i = 2^{N-i} | i = 0, 1, 2, ...M\}, \tag{2.12}$$

where N defines the initial threshold, while M defines the number of threshold we use in the set. Adam [29] is used in both algorithms for both λ and x optimization.

The results are shown in Fig. 2.1. The Lagrangian method is over-conservative, and stuck in a point where it is safe but the doesn't reach the lowest function value of $f(x)$. In constrast, the self-paced method finds the optima without the constraint first, and then gradually adjusts the parameters to satisfy the constraint.

2.5.2 Safety Gym Experiment

We evaluate our algorithm on the safety RL benchmark Safety Gym [46]. The constraints for safety gym environment is defined as a limited tolerance of interacting with hazards and obstacles. Defining different cost threshold amounts to defining different budget for unsafe actions. We compare PPO-Lagrangian and the proposed self-paced PPO-Lagrangian method in this experiment. We use 5 seeds for training for each task.

The experiment results are shown in Fig. 2.2. SP PPO-Lagrangian has better performances in terms of the final return with similar final cost as PPO-Lagrangian. At the beginning of training, the return for SP PPO-Lagrangian increases much faster than PPO-Lagrangian and the cost reduces more slowly. This indicates that the policy is doing exploration more aggressively with the self-paced learning. In Point Push2, SP PPO-Lagrangian does not have a significant improvement over PPO-Lagrangian. We hypothesize that it is more challenging than other environments to get high returns even without considering the cost constraints.

2.6 Conclusion

In this work, we propose Self-Paced Safe Reinforcement Learning (SPSRL). We predefine a set of safety threshold indicating different difficulties. The agent is allowed to choose a suitable threshold for training based on its current performance. In this way the agent learns to optimize for the reward aggressively first and gradually finetunes to satisfy the safety constraints. We demonstrate the benefits of our algorithm in a toy example and the Safety Gym environments.

In the future, we aim to evaluate our method in more diverse settings and combine it with other safe RL algorithms besides PPO-Lagrangian. We are also interested in extending this work by experimenting with other ways of incorporating curriculum to study the role of curriculum in safe RL more systematically.

Chapter 3

Reinforcement Learning in a Safety-Embedded MDP with Trajectory Optimization

3.1 Introduction

[1] Despite the success of reinforcement learning (RL) in sequential decision-making problems [2, 7, 30, 33, 38, 43], it is still challenging to deploy these algorithms in real-world robotics systems. One of the major concerns is that it is hard to ensure safety during deployment, especially for safe-critical applications such as autonomous driving.

A common way of incorporating safety considerations in RL is to frame the task as a Constrained Markov Decision Process (CMDP) in which the goal is to maximize the reward while satisfying the safety constraints given by a cost function and a maximum cost threshold [1, 5, 8, 46]. However, most methods that optimize over a CMDP struggle with the trade-off between maximizing the reward and satisfying the safety constraints, which leads to unstable training, poor performance, or unsafe policies at convergence [46].

In this work, we propose a method to improve safe RL by combining RL with

[1]Videos of our work are shown on our website: https://sites.google.com/view/safemdp

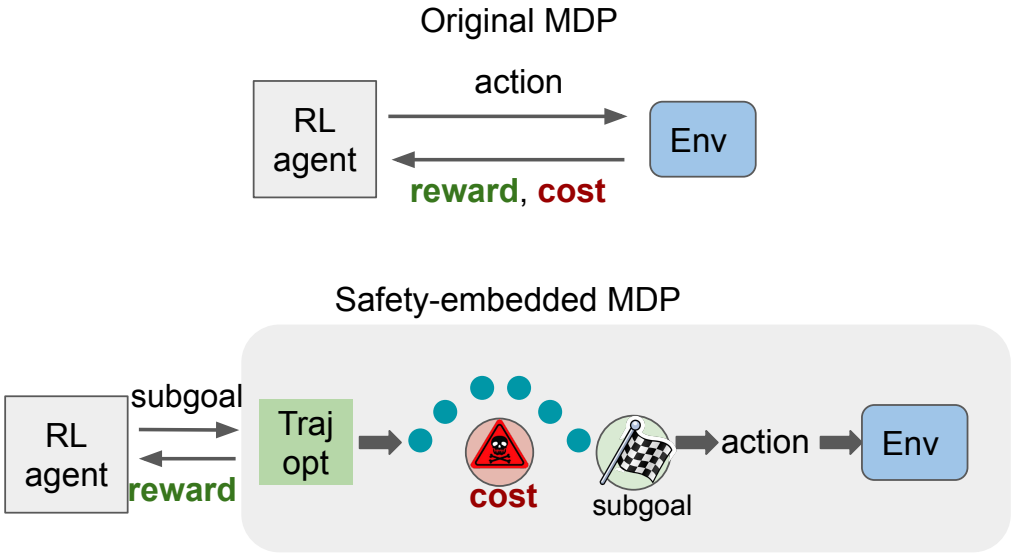


Figure 3.1: Compared to previous methods, in which the RL agent optimizes the reward and safety constraints simultaneously, our method operates in a modified MDP. The modified MDP is embedded with a trajectory optimizer to ensure constraint satisfaction. The RL agent outputs the subgoal for the safe trajectory optimizer and hence the RL agent only needs to optimize explicitly for the reward, leading to much better performance and fewer safety violations.

trajectory optimization for problems in which the safety constraints are defined based on obstacle avoidance. In our method, an RL agent operates on a modified Markov Decision Process (MDP), which is embedded with a trajectory optimization algorithm to ensure safety. Specifically, the RL policy outputs a subgoal; then a trajectory optimizer plans a safe trajectory to reach the subgoal. Because the trajectory optimizer is constrained to only generate safe trajectories, this defines a new high-level action space for the RL agent in which the safety constraints are embedded into the transition dynamics of the MDP. Hence, the RL agent can optimize an unconstrained objective, leading to much faster and more stable training, better performance, and guarantees of safety. The framework of our method is shown in Fig. 3.1

Our method is able to take advantage of the benefits of both RL and trajectory

optimization. The use of trajectory optimization can ensure safety, while the RL algorithm can be used to find an optimal policy, conditioned on the output of the trajectory optimizer. The trajectories returned by the trajectory optimizer are guaranteed to be safe, assuming that the robot is initialized in a safe configuration with 0 initial velocity (although in practice, there can be safety violations due to errors in perception or control).

We demonstrate in our experiments that our method has 0 safety errors during inference in both simulation and in the real world. We evaluate our method on a box-pushing task in the Safety Gym [46] simulation benchmark and show that our method is able to achieve zero safety violations with substantially higher rewards compared to the baseline methods. We also transfer our method to the real world and demonstrate a Franka Panda robotic arm pushing a box around obstacles to reach a goal.

In summary, our contributions include:

- We propose a method to improve safe RL by embedding the safety constraints into an MDP consisting of a trajectory optimizer that is used to plan a safe trajectory.
- The safety of our system with reinforcement learning is guaranteed by the safety of the trajectory optimizer, regardless of the output of the RL agent.
- We demonstrate that our method can achieve **0 safety violations** in contact-rich manipulation tasks in both simulation and in the real world.

3.2 Related Work

3.2.1 Safe Reinforcement Learning

Safe RL is usually framed as a Constrained Markov Decision Process problem of maximizing the return while ensuring the cost is under a certain threshold [5]. One popular approach in safe RL is to convert the safety constraints into a part of the reward objective by using a Lagrangian multiplier [8, 27, 37, 46, 48, 49]. However, these methods generally suffer from training instabilities and do not have safety guarantees. Our method shows better training stability and ensures zero

safety violations. Alternatively, other work ensures safety by constraining the policy behavior or policy class during the update given a safe initial policy [1, 50, 56]. We do not rely on the assumption of having a safe initial policy.

Another line of work in safe RL ensures safety by using shielding functions. The action output of the policy is monitored and corrected by a shielding function to ensure safety [3, 57]. Alshiekh et al. [3] assumes a given shielding function which requires additional domain knowledge. Yu et al. [57] learns a shielding function to remove this assumption. However, they were not able to achieve zero safety violations. We take a similar perspective as this line of work of modifying the action space to ensure safety. A major difference is that we use trajectory optimization in the modified MDP which provides additional structure to the problem and guarantees safety.

3.2.2 Trajectory Optimization

In our method, the trajectory optimizer is used to solve a path-planning problem with obstacle avoidance. Path planning with obstacle avoidance is a canonical problem in motion planning [17, 31, 35]. We choose an optimization-based motion planner in this paper, but it could potentially be replaced by other planning methods. These methods by themselves are usually designed to avoid any collisions which exclude object interactions such as pushing a box. Our method augments the base path planning method to allow a wider range of tasks that may involve object interactions beyond collision-free movement.

Beyond path planning, some of the trajectory optimization methods based on Model Predictive Control (MPC) could be applied to contact-rich tasks such as pushing a box [23, 24, 25]. Unlike our method, these approaches assume a physics model of the environment.

3.2.3 Combine learning with Trajectory Optimization

Trajectory optimization has been combined with learning in various ways. For example, previous work has proposed to use imitation learning to learn the hyperparameters for a motion planner [6, 12, 34] . Several other works have proposed to combine RL and motion planning similar to our work [28, 55]. In contrast, our method combines

RL and trajectory optimization for safety-critical tasks by separating the CMDP objectives into a hierarchical structure.

3.3 Background: Constrained Markov Decision Process

A Constrained Markov Decision Process (CMDP) [5] is formulated as a tuple $(S, A, P, r, \rho, \gamma, c)$ where S is the state space, A is the action space, $P : S \times A \times S \to [0, 1]$ is the transition probability function, r is the reward function, ρ is the distribution of the initial state $\mathbf{s}_0$, γ is the discount factor, and c is the cost function that defines the safety constraint. The training objective of a CMDP is defined as

$$\begin{aligned} \max_{\theta} J_r(\pi_\theta) &:= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)] \\ \text{s.t. } J_c(\pi_\theta) &:= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t c(\mathbf{s}_t, \mathbf{a}_t)] \leq C, \end{aligned} \tag{3.1}$$

where C is a cost threshold, π_θ is the policy with parameters θ, and $\tau := \{s_0, a_0, s_1, a_1, ...\}$ denotes a trajectory where actions are sampled from the policy π_θ and states are sampled from the transition function.

3.4 Problem Statement and Assumptions

In this work, we focus on the specific problem in which the safety constraints are defined by obstacles that we want to avoid. Specifically, all constraints are of the form:

$$||\mathbf{s}_{t,x} - \mathbf{x}_j^{obs}|| > \epsilon \quad \forall t, j \tag{3.2}$$

where $\mathbf{s}_{t,x}$ is the location of the robot at the tth time step, $\mathbf{x}_j^{obs}$ is the location of the jth obstacle, and ϵ is a safety margin that determines how far the robot needs to stay away from the obstacles. We also assume access to a sensor that allows us to obtain noisy measurements of the location of the obstacles $\mathbf{x}_j^{obs}$. Further, we focus on a stricter version of the CMDP problem in which the cost threshold $C = 0$, which

corresponds to a constraint of not colliding with the obstacles at all, rather than having a limited safety budget for the robot to occasionally collide with the obstacles. Finally, in our environments, some objects might be acceptable to interact with (such as a box that we want the robot to push) and others will occur a collision cost $c(\mathbf{s}_t, \mathbf{a}_t)$; we assume that the robot knows the type of each object and whether there will be a collision cost for interacting with that object. Despite these assumptions, solving such a CMDP is still a challenging task because of the difficulty in balancing the objective with the constraints, the difficulty of long-horizon reasoning, and the difficulty of reasoning about contacts such as safely pushing a box to a goal.

3.5 Method: Safe Contact-rich Hierarchical Reinforcement Learning with Trajectory Optimization

Overview: We propose a method that combines reinforcement learning and trajectory optimization in a hierarchical structure. Instead of training an RL policy in the original action space, we propose to learn the policy in a modified action space defined by the parameters of a trajectory optimizer. Using the parameters from the policy output, the trajectory optimizer will plan a path for the agent while taking into account the safety constraints. The optimized trajectory will then be sent to a trajectory-following module that chooses robot actions to follow the path.

In summary, our method consists of three layers: A high-level RL agent that outputs parameters for the trajectory optimizer, a mid-level trajectory optimizer that outputs a safe trajectory, and a low-level trajectory-following module that executes the trajectory. A summary of our method is shown in Fig. 3.2 and the pseudo-code is shown in Alg. 2. We will explain below how this approach leads to a guarantee of safe performance (under certain assumptions) while also enabling our agent to learn contact-rich policies.

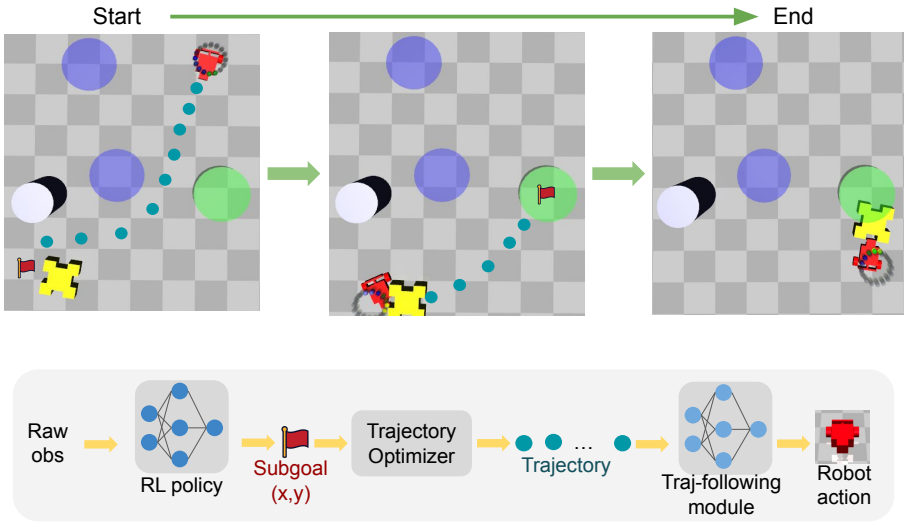


Figure 3.2: An illustration of our method is shown above. In the Push task from the Safety Gym Benchmark, the objective of the agent (red) is to push the box (yellow) to a goal (green) while avoiding obstacles (purple). We propose a method to embed safety constraints into the low-level trajectory optimizer to generate a safe trajectory (the dark green dots in the figure) leading toward the subgoal. The high-level RL policy outputs a subgoal (the red flag in the figure). The RL policy reasons about rich contacts with the box and updates the subgoal output to achieve the task.

Algorithm 2 Reinforcement Learning in a Safety-Embedded MDP with Trajectory Optimization

Require: RL policy output interval k, goal-following agent π_ϕ.
  Initialize the replay buffer D, high-level RL policy π_θ.
  for each episode **do**
    for each environment step **do**
      Select subgoal: $\mathbf{a}'_t \sim \pi_\theta(\mathbf{s}_t)$.
      Input the subgoal $\mathbf{a}'_t$ to the trajectory optimizer to obtain a safe trajectory: $\mathbf{X} \leftarrow \text{TrajOpt}(\mathbf{s}_t, \mathbf{a}'_t)$.
      Initialize cumulative reward $r_t \leftarrow 0$.
      Initialize the initial state $\mathbf{s}_{t,i} \leftarrow \mathbf{s}_t$.
      for $i = 1$ to k **do**
        Follow the traj: $\mathbf{a}_{t,i} \leftarrow \text{TrajFollow}(\mathbf{s}_{t,i}, \mathbf{X}, \pi_\phi)$.
        Execute $\mathbf{a}_{t,i}$ and observe reward $r_{t,i}$ and $\mathbf{s}_{t,i}$.
        Sum the reward $r_t \leftarrow r_t + r_{t,i}$.
      end for
      Save the final state: $\mathbf{s}_{t+1} \leftarrow \mathbf{s}_{t,k}$.
      Store the transition $(\mathbf{s}_t, \mathbf{a}'_t, \mathbf{s}_{t+1}, r_t)$ into D.
      Update the policy π_θ on data from D.
    end for
  end for

3.5.1 Safety-Embedded Markov Decision Process

In order to optimize the CMDP (Section 3.3), we propose to train an RL policy over a Safety-Embedded Markov Decision Process (SEMDP). In the SEMDP, the state space S, the initial state distribution ρ, and the discount factor γ remain the same as in the original MDP. We define a modified action space A' to be a set of parameters that will be input into the trajectory optimizer. Specifically, we use a subgoal position for the agent as the action space of the RL agent in our experiments, which is the desired location of the "root node" of the agent. The root node is processed as the center of the robot by default in the simulator. (please see Appendix B.2.1 for the definition of the "root node"). The trajectory optimizer (described below) will then find a safe trajectory to reach the subgoal.

Based on this new action space A', we define a new transition function $P' : S \times A' \times S \to [0, 1]$ which depends on the trajectory optimizer and the trajectory following module. Given the current state $\mathbf{s}_t$ and action $\mathbf{a}'_t \in A'$, the trajectory optimizer (details below) will plan a safe trajectory to reach the subgoal. The trajectory-following module will then take k actions in the original MDP to follow the trajectory. Thus, the state $\mathbf{s}_{t+1}$ that is reached after taking action $\mathbf{a}'_t$ depends on the operation of the trajectory optimizer and the trajectory-following module. From the perspective of the RL agent, this transition is recorded as the tuple $(\mathbf{s}_t, \mathbf{a}'_t, \mathbf{s}_{t+1})$. Because the SEMDP is operating over k time steps in the original MDP, the reward function $r'(\mathbf{s}_t, \mathbf{a}_t)$ is modified to be the accumulated reward over k steps.

Importantly, the SEMDP does not need to be a CMDP, e.g. it does not include an explicit cost constraint. This is because the cost is accounted for in the modified action space, which uses a trajectory optimizer to find a safe trajectory to reach the subgoal. If the trajectory optimizer finds a safe trajectory and if the trajectory-following agent correctly follows the trajectory, then all states visited by the agent will be safe (i.e. they will have 0 costs). Thus, in the SEMDP, the RL agent does not need to explicitly optimize for the cost, because the cost is implicitly accounted for in the modified action space. As we will see, this change makes the RL optimization significantly easier. Because the SEMDP does not need an explicit cost constraint, we train the RL agent in the SEMDP with a standard method for model-free reinforcement learning (details in Section 3.5.4).

3.5.2 Trajectory Optimizer

The goal of the trajectory optimizer is to find a safe and feasible trajectory reaching the subgoal $\mathbf{a}'_t$. First, the trajectory must be kinematically feasible for the robot to follow using the trajectory-following module. In this work we assume that a sufficiently smooth trajectory can be followed by the robot; we leave for future work to incorporate a robot-specific transition feasibility function. Second, the trajectory must be safe in that it should not collide with any obstacles. In this work, we discretize the trajectory into N waypoints, denoted as $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, in which $\mathbf{x}_i$ defines the position of the "root node" of the agent. We also define the velocity at each waypoint as $\mathbf{V} := \{\mathbf{v}_1, \mathbf{v}_1, ..., \mathbf{v}_N\}$.

Mathematically, we define the following constrained optimization problem for the trajectory optimizer:

$$\begin{aligned} & \min_{\mathbf{X},\mathbf{V}} f_{goal}(\mathbf{X}, \mathbf{a}'_t) \\ \text{s.t.} \quad & h_{init}(\mathbf{X}, \mathbf{s}_{t,x}) \leq \sigma_{init} \\ & h_{smooth}(\mathbf{X}, \mathbf{V}) \leq \sigma_{smooth} \\ & \sum_{i,j} h_{cost}(\mathbf{x}_i, \mathbf{x}_j^{obs}) \leq 0, \end{aligned} \tag{3.3}$$

in which $\mathbf{s}_{t,x}$ is the position of the root node of the agent at time step t and σ_{init} and σ_{smooth} are constants that define the constraint limits. We define each component of this optimization problem below:

1. The optimization objective encourages the final waypoint of the trajectory to align with the subgoal location $\mathbf{a}'_t$ that was output by the RL policy:

$$f_{goal}(\mathbf{X}, \mathbf{a}'_t) := ||\mathbf{x}_N - \mathbf{a}'_t||^2 \tag{3.4}$$

   in which $|| \cdot ||$ denotes the $L2$ distance. Note that subgoal reaching is in the objective of this optimization but is not enforced as a constraint. Thus, occasionally the trajectory optimizer will fail to find a trajectory that reaches the subgoal $\mathbf{a}'_t$ in order to satisfy the safety constraints.

2. The first constraint enforces that the initial waypoint needs to be located at the current position of the root node of the robot, $\mathbf{s}_{t,x}$. The corresponding cost

function is defined as:

$$h_{init}(\mathbf{X}, s_{t,x}) := ||\mathbf{x}_1 - \mathbf{s}_{t,x}||^2. \tag{3.5}$$

3. The second constraint enforces that the trajectory must be smooth, in order to ensure kinematic feasibility. We optimize for the location of the waypoints and the corresponding velocity at these waypoints. Non-smooth locations and changes in the velocities are penalized. Specifically, the smoothness cost is defined as the following equation:

$$h_{smooth}(\mathbf{X}, \mathbf{V}) := \sum_{i=1}^{N-1} \left\| \begin{array}{c} \mathbf{x}_{i+1} - \mathbf{x}_i - \mathbf{v}_i \Delta t \\ \mathbf{v}_{i+1} - \mathbf{v}_i \end{array} \right\|_K^2 \tag{3.6}$$

in which $||\cdot||_K$ is the Mahalanobis distance with a metric given by K and Δt is the time interval between two adjacent waypoints; this smoothness cost is derived from a constant velocity GP prior with an identity cost-weight; see prior work [9, 40] for details.

4. The last set of constraints enforces that the trajectory needs to avoid collisions with obstacles. The cost of the ith the waypoint with the jth obstacle is defined as:

$$h_{cost}(\mathbf{x}_i, \mathbf{x}_j^{obs}) := \left\{ \begin{array}{cc} 0 & \text{if } d_{i,j} > \epsilon' \\ (\epsilon' - d_{i,j})^2 & \text{otherwise} \end{array} \right. , \tag{3.7}$$

in which $\mathbf{x}_j^{obs}$ denotes the location of the jth obstacle, $d_{i,j} := ||\mathbf{x}_i - \mathbf{x}_j^{obs}||$ denotes the distance between the ith waypoint and the jth obstacle, and ϵ' denotes a distance threshold. We choose ϵ' such that $\epsilon' \geq \epsilon$, in which ϵ is the distance threshold specified by the problem definition in Equation 3.2. We choose $\epsilon' \geq \epsilon$ to account for perception errors and errors in the trajectory-following agent.

We solve the constrained optimization problem in Equation 3.3 using the method of dual descent:

$$\max_{\boldsymbol{\lambda} \geq 0} \min_{\mathbf{X}, \mathbf{V}} f_{goal}(\mathbf{X}, \mathbf{a}'_t) + \boldsymbol{\lambda}^T \left(\begin{array}{c} h_{init}(\mathbf{X}, s_{t,x}) - \sigma_{init} \\ h_{smooth}(\mathbf{X}, \mathbf{V}) - \sigma_{smooth} \\ \sum_{i,j} h_{cost}(\mathbf{x}_i, \mathbf{x}_j^{obs}) \end{array} \right). \tag{3.8}$$

The inner loop is optimized using a trajectory optimizer; in practice, we use the Levenberg-Marquardt algorithm [54] implemented in Theseus [44]. The outer loop is optimized using gradient descent on $\boldsymbol{\lambda}$. Please see Appendix B.2.2 for more implementation details about the trajectory optimizer.

We know that a feasible solution to the optimization problem exists as long as the robot's initial state $\mathbf{s}_{t,x}$ is feasible (e.g. greater than or equal to ϵ' away from any obstacle). In such a case, a stationary trajectory (all waypoints located at $\mathbf{s}_{t,x}$) would satisfy all of the constraints. We initialize the optimization for the trajectory optimizer at the stationary trajectory (all waypoints located at $\mathbf{s}_{t,x}$). In this way, we guarantee that our optimization will always return a feasible trajectory because we can always return the stationary trajectory if no other feasible trajectory is found. Thus, we guarantee that the trajectory returned by the trajectory optimizer will be safe, meaning that no waypoints will be within ϵ' of any obstacle.

However, in practice, the executed trajectory might not be perfectly safe, due to perception errors or errors with the trajectory-following module. Specifically, there might be errors in practice due to a number of factors:

1. Localization errors in estimating the robot position $\mathbf{s}_{t,x}$
2. Perception errors in estimating the locations of the obstacles $\mathbf{x}_j^{obs}$
3. Errors in following the planned trajectory
4. Discretization errors: since the planned trajectory is a discrete set of waypoints $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, it is possible that the agent will encounter an unsafe state in between the trajectory waypoints that was not accounted for in the trajectory optimization.
5. If the robot dynamics is not quasi-static, then after the first trajectory optimization, the robot velocity will be non-zero, so a perfectly stationary trajectory might not be feasible; hence it is possible that no safe trajectory exists from the robot's current state. This can be alleviated if the trajectory-following module finds an action sequence to bring the final velocity to 0 at the end of each execution.

To account for these inaccuracies, we set $\epsilon' > \epsilon$; nonetheless, the strength of our framework is that these are the only potential sources of unsafety; the waypoints returned by the trajectory optimizer are guaranteed to not intersect with obstacles,

as stated above, regardless of the output of the RL agent.

In practice, since the Lagrangian method adds computational overhead, we use a fixed $\boldsymbol{\lambda}$ during training, which speeds up training time and also encourages exploration (although this leads to some safety violations during training). At test time, we update $\boldsymbol{\lambda}$ as mentioned previously, which ensures safety at test time.

3.5.3 Trajectory-Following Module

The trajectory optimizer outputs a set of waypoint locations; we ignore the velocities output by the trajectory optimizer in the trajectory-following module, since their purpose was only to define the smoothness cost $h_{smooth}(\mathbf{X}, \mathbf{V})$. Additionally, we will demonstrate in the experiments below that controlling the robot to follow the positions of waypoints is enough to solve the task. Next, we use a trajectory-following module that operates in the original robot action space to track the waypoints. Given the trajectory $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, the trajectory-following module selects the next waypoint $\mathbf{x}_i$ and inputs the waypoint to the goal-following agent to generate low-level robot actions $a_t = \pi_\phi(\mathbf{s}_t, \mathbf{x}_i)$.

Our overall system is agnostic to the form of the goal-following agent; in our experiments, we train the goal-following agent using reinforcement learning in an obstacle-free environment with only the robot and a randomly sampled goal. The agent is goal-conditioned $\pi_\phi(\mathbf{s}_t, \mathbf{g})$ and is trained to reach a goal $\mathbf{g}$ that is randomly sampled around the robot. Specifically, the goal $\mathbf{x}_i$ is sampled uniformly from a range of d_{min} to $2d_{min}$ away from the robot to match the input distribution during inference, in which d_{min} is a distance threshold. The reward function used for training the goal-following agent is defined as the change in distance between the robot and the goal:

$$r_t^g := ||\mathbf{s}_{t-1,x} - \mathbf{g}|| - ||\mathbf{s}_{t,x} - \mathbf{g}||. \tag{3.9}$$

More implementation details about the trajectory-following module are in Appendix B.2.2.

3.5.4 Implementation details

RL policy: We use Soft Actor-Critic (SAC) [21] to train the high-level policy. The actor network and the critic network are three-layer neural networks with a hidden size of 256. The action space is defined as $[-2, 2]$, which defines the farthest possible subgoal output at the current time step relative to the robot. The learning rate for training is set to be 3e$-$4. The agent interacts with the environment for 1e7 time steps to ensure it has fully converged.

Trajectory optimizer: The distance threshold ϵ' is set to be $0.5m$ during training for all of the robots. The number of waypoints is set to be 30. The maximum time step for optimizing the trajectory is set to be 10 steps during training. If the robot is currently within ϵ' of some obstacle, we would manually set $\boldsymbol{\lambda}$ to be the maximum value $\boldsymbol{\lambda}_{max}$ to encourage the robot to escape the unsafe region as quickly as possible.

Trajectory-following module: SAC is used to train the goal-following agent. The actor network and the critic network are three-layer neural networks with a hidden size of 256. d_{min} is set to be $0.2m$. The agent interacts with the environment for 1e6 time steps to ensure it has fully converged.

3.6 Experiments

In this section, we evaluate our method on Safety Gym simulation benchmarks [46]. We also transfer the policy to a real-world task of pushing a box around obstacles to a goal.

3.6.1 Safety Gym Setup

Safety Gym [46] is a set of benchmark environments which can be used to evaluate methods under a CMDP framework. In our experiments, we focus on the "Push" tasks of Safety Gym, in which the robot has to push a box towards a goal and avoid obstacles. The Push tasks require reasoning about rich contacts between the robot and the environment, while also reasoning about safety; this environment is challenging for previous methods. In previous work on Safety Gym, a cost threshold of

25 was used [46]; in contrast, we use a stricter cost threshold of 0 in our experiments, to not allow any safety violations.

The following types of objects exist in the environments for the Push tasks:

1. **Goal** indicates where the robot needs to reach. As long as the robot is within a specific range of the goal, the task is considered successful.
2. **Hazard** denotes circular regions that the robot should not enter. It is a virtual region and does not interact with the robot. The robot entering such a region will incur the cost. It is defined as obstacles in our method.
3. **Pillar** is a fixed cylinder; contact with the pillar will not incur any cost. Though contacting with pillar might now incur cost, the robot might get stuck. Thus, it is also defined as an obstacle in our method.
4. **Box** is an object that the robot can (and must) interact with: the goal is for the agent to push the box towards the goal. If the box enters the hazard regions, it will not incur any cost.

The reward function for the "Push" task is defined as the change in the distance between the robot and the box and between the box and the goal:

$$r_t^p := ||\mathbf{x}_{t-1}^b - \mathbf{g}|| - ||\mathbf{x}_t^b - \mathbf{g}|| + ||\mathbf{s}_{t-1,x} - \mathbf{x}_{t-1}^b|| - ||\mathbf{s}_{t,x} - \mathbf{x}_t^b||, \tag{3.10}$$

in which $\mathbf{x}_t^b$ denotes the location of the center of the box at the t th step, and $\mathbf{g}$ denotes the location of the final goal that the box needs to reach. The cost function is defined as:

$$c_t^p := \mathbb{1}(\min_j ||\mathbf{s}_{t,x} - \mathbf{x}_j^{obs}|| < \epsilon), \tag{3.11}$$

in which $\mathbb{1}$ is the indicator function, and ϵ is a distance threshold. The cost function will return 1 if the "root" of the robot is within ϵ of any obstacle. The observation space of the Safety Gym environment consists of LiDAR observation and robot states. The LiDAR observation returns an approximate position of the objects, due to the limited LiDAR resolution. The robot states consist of robot velocity, acceleration, and orientation.

There are several robot morphologies in the Safety Gym. We use the Point agent and the Car agent. The Point agent is a robot with one actuator for turning and another actuator for moving forward or backward. The Car agent is a robot with two

independently driven wheels. In addition, we also created another agent called Mass, which is an omnidirectional agent; the action space of the Mass agent is defined as a delta movement of the agent position.

We compare our method to the state-of-the-art safe RL methods: CPO [1], PPO-Lagrangian (PPO Lag), and TRPO-Lagrangian [46] (TRPO Lag). During training, four seeds are used for each method. We also evaluate the performance of the policy every 20 episodes. During the evaluation, we choose the mean action from the policy instead of sampling from the policy distribution.

3.6.2 Safety Gym Results

The results are shown in Fig. 3.3. We apply the Savitzky-Golay filter [47] to smooth the curve for better visualization. As shown, our method achieves a much higher reward than the baselines with very little incurred cost. As noted previously, we use a fixed $\boldsymbol{\lambda}$ during training to speed up computation and to encourage exploration, which leads to some safety violations during training; at test time, we adjust $\boldsymbol{\lambda}$ to optimize Equation 3.8, leading to safe trajectories.

After training is complete, we evaluate the policy at the final episode. For our method, each agent has interacted with the environment for 1e7 time steps. For the baselines, each agent has interacted with the environment for at least 4e7 time steps. The baseline methods have more time steps to ensure convergence. Each policy is evaluated for 50 episodes and the average results are shown in Table 3.1. The number of safety errors from our method is reduced to zero because we adjust $\boldsymbol{\lambda}$, unlike in training when $\boldsymbol{\lambda}$ is held fixed.

In prior work on Safety Gym, a cost threshold of 25 was used [46]; in our experiments, we use a stricter cost threshold of 0. This leads to significantly worse performance for the Lagrangian methods, which are unable to achieve a reasonable reward due to training instability.

3.6.3 Ablations and Additional Analysis

In this section, we perform additional ablation experiments to understand the reason behind our method's high performance.

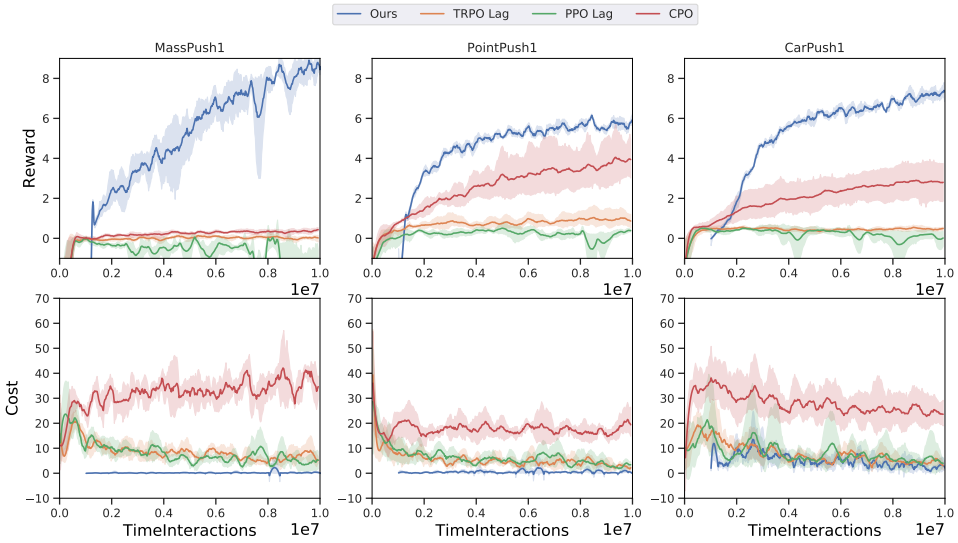


Figure 3.3: Training curves of our method compared to the baseline methods. The curves have been smoothed for better visualization. Four different seeds are used for each method. The shadow region denotes the standard error of different seeds. Our method starts from 1e6 steps instead of 0 to denote the training of the goal-reaching policy. Our method achieves a lower cost than the baselines. It still incurs some cost during training because, during training time, we are using a fixed Lagrangian parameter for computation reasons and to encourage exploration.

Table 3.1: Evaluation results of the final policy, using an adaptive Lagrangian parameter for our method. See text for details. Our method was trained for $1e7$ environment interaction steps and each of the baseline methods were trained for $6e7$ environment interaction steps.

		Ours	CPO	PPO Lag	TRPO Lag
MassPush1	reward	**4.31**	0.78	-2.26	-0.39
	cost	**0.00**	31.48	2.6	0.05
PointPush1	reward	**5.69**	5.47	-5.61	0.68
	cost	**0.00**	16.45	6.61	0.45
CarPush1	reward	**4.57**	4.4	-2.34	0.45
	cost	**0.00**	30.35	3.17	1.46
		PPO Lag + SAC	SAC + PPO Lag	Ours w/ gt	
MassPush1	reward		14.62	10.00	
	cost		40.25	0.00	
PointPush1	reward	0.16	-0.87	5.70	
	cost	1.41	24.16	0.00	
CarPush1	reward	0.15	0.18	5.33	
	cost	5.58	6.75	0.00	

How much of our improvement over the baselines is attributed to using a trajectory-following module? To understand this, we evaluate our method in the "MassPush1" environment which uses a Mass agent, in which the action space is defined as a delta movement of the agent position. As such, we do not need to use a trajectory-following module for this agent, either in our method or in any of the baselines. As shown in the "MassPush1" experiments in Fig. 3.3 and Table 3.1, our method still significantly outperforms the baselines. This demonstrates that the benefits of our method are not from using a trajectory-following module itself. We believe that the benefits come from training an RL agent in a Safety-Embedded MDP defined by a safe trajectory optimizer.

As an additional experiment to understand the effects of a trajectory-following module, we modify the baselines to incorporate a goal-reaching low-level agent. Specifically, we train a goal-reaching agent with SAC to reach arbitrary subgoals sampled in the environment, and then we use PPO Lagrangian to train a high-level policy that outputs a (hopefully safe) subgoal. The results of this experiment can be found in Table 3.1, referred to as "PPO Lag + SAC." The results for the Mass agent are left blank since the Mass agent does not require a low-level goal-reaching policy.

As can be seen, "PPO Lag + SAC" performs poorly, incurring a low reward and many safety violations. Though "PPO Lag + SAC" performs slightly better than the PPO Lagrangian method, there is still a huge performance gap compared to our method. This demonstrates that the benefits of our method do not come directly from the trajectory-following module; incorporating such a module into the baselines still leads to poor performance.

Do we need a trajectory optimizer? In this ablation, we replace the safe trajectory optimizer in our method with a low-level policy trained with PPO Lagrangian to find a safe trajectory to reach a subgoal. Instead of using a trajectory optimizer, we use PPO Lagrangian to train a low-level policy in the same environment as the Push task to (hopefully safely) reach a randomly sampled goal (rather than being trained to push the box to the goal). Then the high-level RL policy is trained with SAC, the same as in our method, to output subgoals. For each subgoal output by the RL policy, the low-level goal-reaching policy (now trained with PPO Lagrangian) will try to (safely) reach the subgoal for k time steps. The intention of this experiment is to be as similar to our method as possible but replacing the trajectory optimizer with a low-level goal-reaching agent trained with safe RL. The results of this experiment can be found in Table 3.1, referred to as "SAC + PPO Lag." As can be seen, this method also performs poorly, demonstrating that a safe trajectory optimizer is needed to ensure safety; training a cost-aware low-level agent with PPO Lagrangian is not sufficient to obtain safety.

How much is our performance affected by perceptual errors? As noted, some of the errors in our system come from perceptual errors, in which the location of the obstacles is perturbed by some noise in the Safety Gym. To measure the effect of these errors, we perform an experiment in which we allow our trajectory optimizer to have access to the ground-truth location of the obstacles, while the RL agent still takes in the noisy LiDAR observations as input. The results can be found in Table 3.1 and it is denoted as "Ours w/ gt". As shown, the performances for all the environments have increased. The MassPush1 task has the greatest improvement in reward by switching to ground-truth locations. This is because perception errors are the major source for errors for the Mass agent.

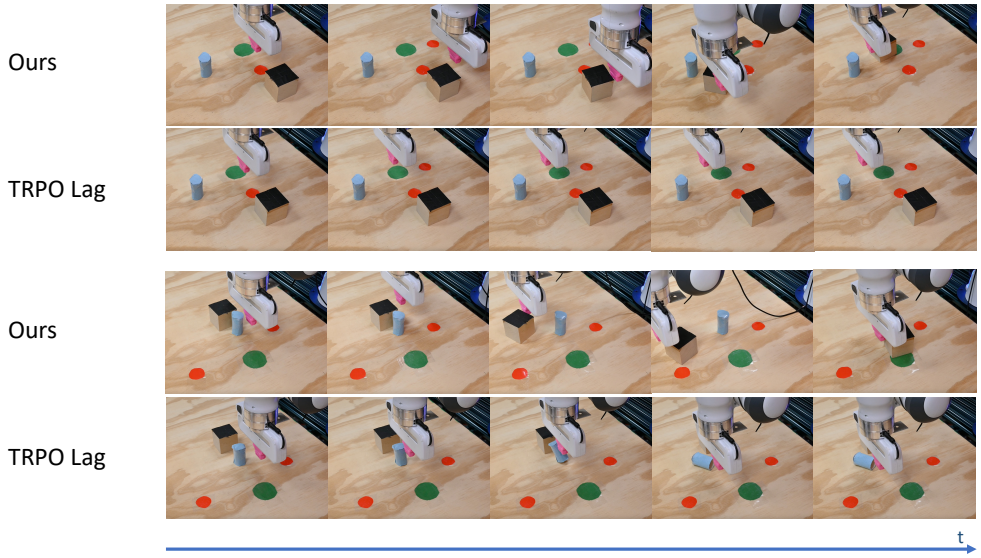


Figure 3.4: We set up a real-robot environment similar to the Safety Gym Push task. The fingertip of the Franka robot (pink) is used to push the box (black) towards the goal (green). It needs to avoid hazards (red) and avoid getting stuck at the pillar (blue). Each row shows five frames of a single episode. We compare our method with TRPO Lagrangian, which has the best performance among the baselines based on the simulation experiments.

3.6.4 Real-Robot Experiments

We use a real-robot version of the Push task for evaluation, using a Franka Panda gripper. As shown in Fig. 3.4, the fingertip of the gripper moves in a plane to push the box towards the goal (the large green circle). As in the simulation setup. the robot also needs to avoid hazards (the small red circle) and try not to get stuck by the pillar (blue). In order to match the observation distribution seen during training, we convert the location of the objects into LiDAR readings in a similar format to the Safety Gym Pseudo LiDAR.

In this experiment, we first train the policy in simulation. We modify the simulation environment to match the setup in the real world, e.g., we change the size of objects and the shape of the robot in the simulation. Then we directly transfer the policy to the real robot without any finetuning. An episode is considered successful if the robot is able to push the box into the green region within 60 time steps.

We compare our method with TRPO Lagrangian. We evaluate each method with

Table 3.2: Results of the real robot experiment. Our method achieves a higher success rate and reward than the baseline. Both methods have 0 costs in the real world.

Method	Succ rate	reward	cost
Ours	**8/10**	**1.33**	**0**
TRPO Lag	0/10	0.22	**0**

10 different layouts; evaluating each layout with both our method and the baseline. The results are shown in Table 3.2. Both methods are safe in the real world, but our method has a much higher success rate and reward. The reasons for the failure of our method include timing out or getting stuck around the obstacles. For the TRPO Lagrangian baseline, the robot is able to move towards the box but is not successful in pushing the box to the goal, which also matches its performance in simulation.

Chapter 4

Conclusions

In this thesis, we propose two methods that could improve safety during the deployment of autonomous systems.

1. In Chapter 2, We propose a hierarchical framework, in which the RL agent optimizes the training objective in a modified MDP which is embedded with a trajectory optimization algorithm to ensure safety. We test our method on Safety Gym benchmarks and a real-robot pushing task, demonstrating better performance than the baselines in terms of both rewards and costs.

   Our framework is not constrained to the trajectory optimization algorithm that we use in our experiments and the output of the RL agent is not constrained to be subgoals. Alternatively, the RL agent can output any parameters that define the objective for the trajectory optimizer, and the trajectory optimizer can take any form as long as it is compatible with the output of the RL policy. We believe our work will contribute to the field of robotics by demonstrating the importance of combining RL and trajectory optimization in safety-constrained optimization tasks.

2. In Chapter 3, we propose a hierarchical framework, in which the RL agent optimizes the training objective in a modified MDP which is embedded with a trajectory optimization algorithm to ensure safety. We test our method on Safety Gym benchmarks and a real-robot pushing task, demonstrating better performance than the baselines in terms of both rewards and costs.

Our framework is not constrained to the trajectory optimization algorithm that we use in our experiments and the output of the RL agent is not constrained to be subgoals. Alternatively, the RL agent can output any parameters that define the objective for the trajectory optimizer, and the trajectory optimizer can take any form as long as it is compatible with the output of the RL policy. We believe our work will contribute to the field of robotics by demonstrating the importance of combining RL and trajectory optimization in safety-constrained optimization tasks.

Appendix A

Self-paced Safe Reinforcement Learning

A.1 Additional Experiment Results on Safety Gym

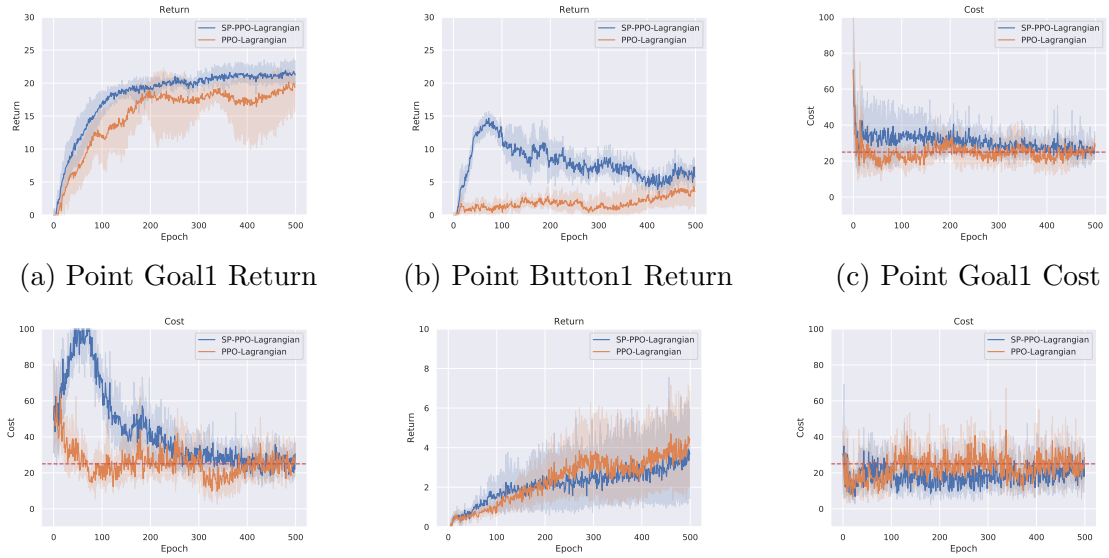
Additional experiment results for safety gym is shown in Fig. A.1.

A.2 Safe Driving Experiment

Safety is one of the critical issues in the autonomous driving domain. We would like to test our constrained optimization algorithm in autonomous driving. We design the Safe Driving environment, which consists of a circle environment and a trajectory environment.

A.2.1 Experiment Setup

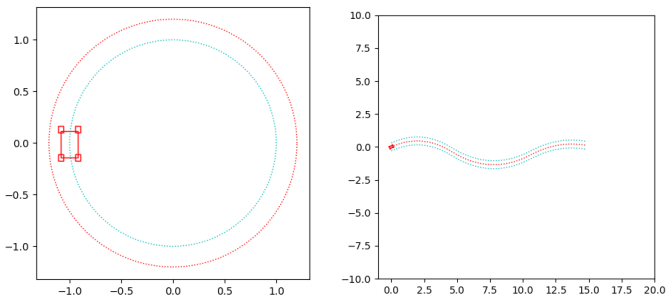
For both environment, we assume a bicycle model as the kinematic model and brush tire model as the dynamics model [45, 52]. We take drifting behaviors into account, which would make the task more challenging. The visualization of the two environments is shown in Fig. A.2:



(a) Point Goal1 Return (b) Point Button1 Return (c) Point Goal1 Cost

(d) Point Button1 Cost (e) Point Push1 Return (f) Point Push1 Cost

Figure A.1: PPO-Lagrangian (orange curve) and SP-PPO-Lagrangian (blue curve) are evaluated in the Safety Gym environment. We choose the point agent and different levels of goal, button, and push tasks. In most of the tasks, our method can converge to a policy with better final returns while has a similar performance as PPO-Lagrangian in terms of satisfying the safety constraints.



(a) Circle Environment (b) Trajectory Environment

Figure A.2: Visualization of the Circle environment and the Trajectory environment. In the Circle environment, the vehicle is required to follow the circle. In the Trajectory environment, the trajectory is randomly generated. The vehicle is required to follow the trajectory. In both environments, we assume a fixed width of the trajectory. Going off the trajectory is considered as dangerous behaviors. We specify different desired velocities in both environments, which defines different levels of difficulties.

For the Circle environment, the vehicle is required to drive along the circle. We assume a fixed width of the trajectory and going off the trajectory is considered as dangerous behaviors. The observation space is defined as the lateral distance to the center of the trajectory, the orientation difference with respect to the tangent of the trajectory, and their corresponding derivatives with respect to time.

For the Trajectory environment, the vehicle is required to drive along the randomly generated trajectory. We assume a fixed width of the trajectory and going off the trajectory is considered as dangerous behaviors. In addition to the observation space in the circle environment, the partial trajectory around the vehicle is also added into the observation space.

For both environments, the reward function is defined as follows:

$$r = d^2 + \omega_1 * (v - v_d)^2 + \omega_2 * \mathbb{I}(|\phi - \phi_d| \geq \frac{\pi}{2}), \tag{A.1}$$

where d is the distance to the center of the trajectory, v is the current velocity, v_d is the desired velocity, ϕ is the orientation of the vehicle, ϕ_d is the desired orientation the vehicle should be at the current time step. w_1 and w_2 are corresponding weights.

The cost is defined as:

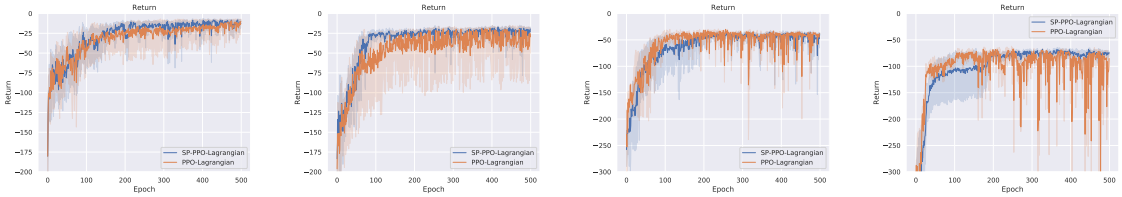
$$c = \max(d - d_0, 0) \tag{A.2}$$

where d_0 is the distance threshold, which usually means the width of the trajectory.

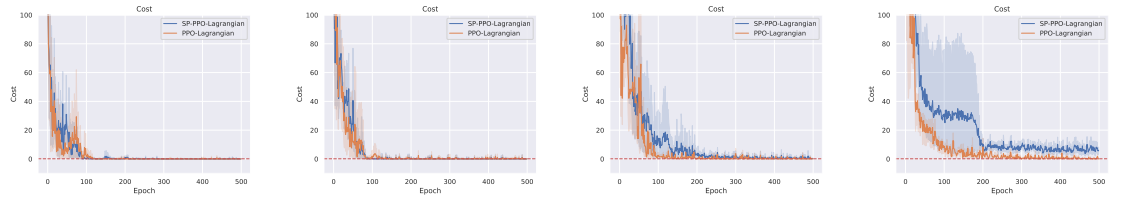
A.2.2 Experiment Results

The results of Circle environment is shown in Fig. A.3. We could tell that SP PPO-Lagrangian has a slightly better return than PPO-Lagrangian. Additionally, the training of SP PPO-Lagrangian is more stable than PPO-Lagrangian. However, if the desired velocity is too high (e.g. $v = 6m/s$), SP PPO-Lagrangian may not converge to a safe policy within limited training steps. This is probably because it has converged to an aggressive policy, it would takes a long time to converge to a safe one.

The results for the Trajectory environment is shown in Fig. A.4. The conclusion here is similar to that in the Circle environment. We could see a greater gap between two methods in terms of return. The larger the velocity is, the larger the gap is.



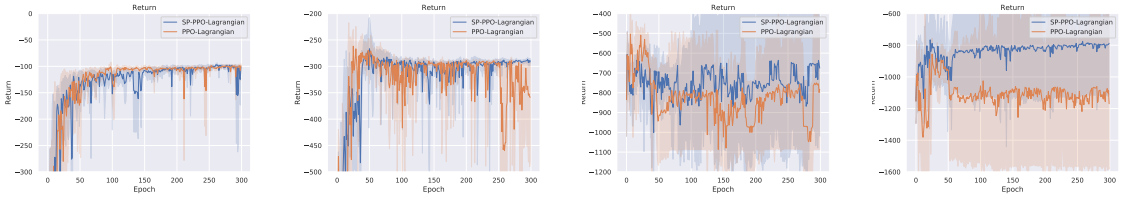
(a) Circle Env (v=2) Return (b) Circle Env (v=3) Return (c) Circle Env (v=4) Return (d) Circle Env (v=5) Return



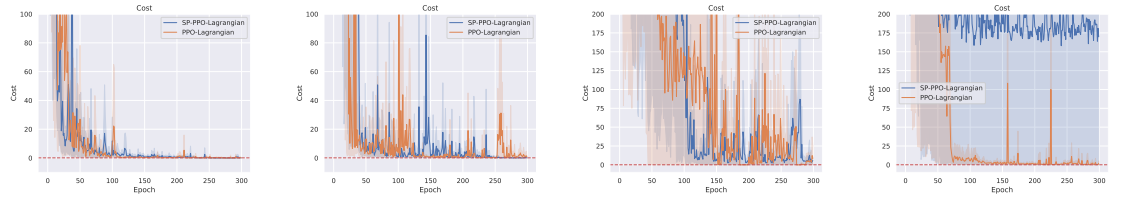
(e) Circle Env (v=2) Cost (f) Circle Env (v=3) Cost (g) Circle Env (v=4) Cost (h) Circle Env (v=5) Cost

Figure A.3: PPO-Lagrangian (orange curve) and SP PPO-Lagrangian (blue curve) are evaluated in the Circle environment. We choose different desired velocities denoting different levels of difficulties. The greater the desired velocity is, the larger the gap between two methods in terms of return. But when desired velocity is too high (e.g. 5 m/s), SP PPO-Lagrangian cannot converge to a safe policy.

The cost for SP PPO-Lagrangian cannot converge to zero if the desired velocity is too high. This is probably because the policy has already learned to be aggressive, it would take a longer time for it to be safe again because it may need to substantially change the strategy.



(a) Trajectory Env (v=2) Return (b) Trajectory Env (v=3) Return (c) Trajectory Env (v=4) Return (d) Trajectory Env (v=5) Return



(e) Trajectory Env (v=2) Cost (f) Trajectory Env (v=3) Cost (g) Trajectory Env (v=4) Cost (h) Trajectory Env (v=5) Cost

Figure A.4: PPO-Lagrangian (orange curve) and SP PPO-Lagrangian (blue curve) are evaluated in the Trajectory environment. The vehicle is required to follow a certain randomly generated trajectory. We choose different desired velocities denoting different levels of difficulties. The greater the desired velocity is, the larger the gap between the two methods in terms of return. But when desired velocity is too high (e.g. 5 m/s), SP PPO-Lagrangian cannot converge to a safe policy.

Appendix B

Safety-Embedded MDP

B.1 Additional Experiments

B.1.1 Adjusting the Trade-off between Reward and Cost

We can adjust the trade-off between the reward and the cost during test time in the proposed method. Specifically, we can adjust the threshold ϵ' defined in Equation 3.2. The results are summarized in Fig. B.1. As a comparison, we also include three baselines: CPO, PPO-Lagrangian, and TRPO-Lagrangian.

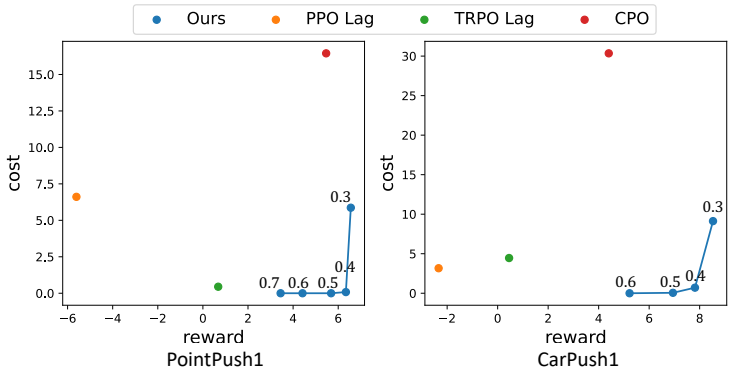


Figure B.1: Trade-off between the reward and the cost for different methods. The blue dots represent our method with different ϵ'. The numbers above the blue dots denote the value of ϵ'.

From Fig. B.1, smaller ϵ' might lead to a less conservative agent with higher

reward and higher cost. Nonetheless, our method has the higher reward and lower cost compared to the baselines across different values of ϵ'.

B.1.2 Our method with fixed Lagrangian

Our method adapts the Lagrangian parameter $\boldsymbol{\lambda}$ over time according to Equation 3.8. We also evaluate an ablation of our method which uses a fixed $\boldsymbol{\lambda}$ ("Ours w/o Lag"). The results are shown in Table B.1. Without adapting the Lagrangian parameter, the policy leads to a higher cost but also obtains a higher reward. With a fixed $\boldsymbol{\lambda}$, the agent may occasionally enter the unsafe regions to achieve a shorter path to the goal. The results demonstrate the necessity of adjusting $\boldsymbol{\lambda}$ in the proposed method to ensure safety.

Table B.1: Evaluation results of the final policy, comparing our method with an adaptive Lagrangian parameter and our method with a fixed Lagrangian parameter

		Ours	Ours w/o Lag
MassPush1	reward	4.31	**8.61**
	cost	**0.00**	0.23
PointPush1	reward	5.69	**5.74**
	cost	**0.00**	0.14
CarPush1	reward	4.57	**5.66**
	cost	**0.00**	0.02

B.1.3 Additional Safety Gym environments

We evaluate our method with an additional task, the "Goal" task in SafetyGym [46]. In the Goal task, the robot itself needs to go to a specific goal and avoid obstacles instead of moving an object to the goal. The results are shown in Fig. B.2. Our method still achieves the lowest cost and a relatively high reward during training.

B.1.4 Analyzing the Ablations

In the main text, we discussed an ablation called "SAC + PPO Lag" in which we train a low-level policy to safely reach goals with PPO Lagrangian, and then we use

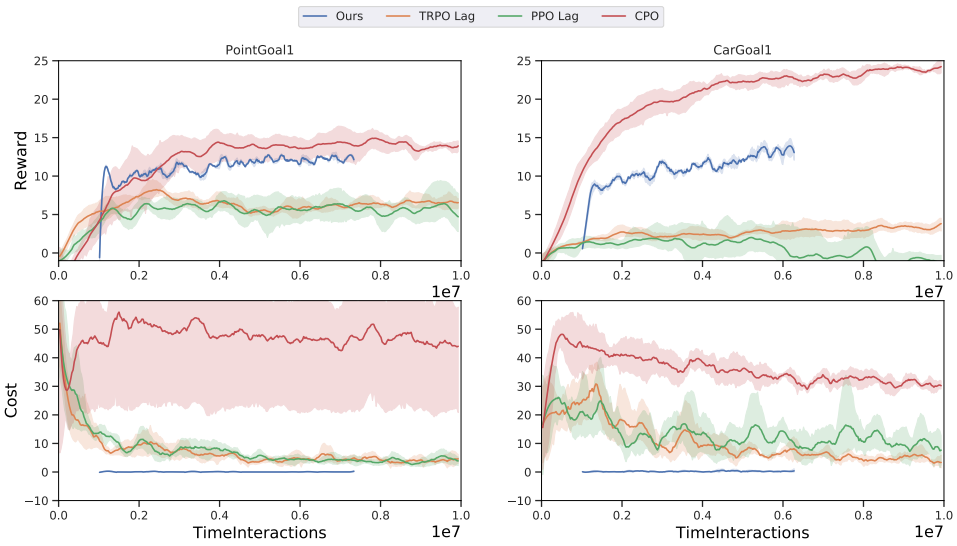


Figure B.2: Training curves of our methods compared to the baseline methods in the Goal environment. The curves have been smoothed for better visualization. Our method starts from 1e6 steps instead of 0 to denote the training of the goal-reaching policy. Our method achieves the lowest cost among all the baseline methods.

SAC to output subgoals (see Section 3.6.3 for further discussion). In this section, we analyze why this method appears to perform so poorly in Table 3.1.

The training curves for the low-level safe goal-reaching policy trained with PPO Lagrangian are shown in Fig. B.3. As can be seen, these policies do not learn to be safe and the cost during training is always above 0. Hence, integrating such a low-level policy with a high-level SAC agent leads to poor overall performance for "SAC + PPO Lag" baseline. This experiment highlights the difficulties of training even a safe short-horizon policy with PPO Lagrangian.

B.1.5 Training Curves of the ablations

The training curve of ablation methods are shown in Fig. B.5 which corresponds to the results in Table 3.1. The ablations have much higher cost than our method during training and significantly higher cost than our method during inference (in which our method obtains 0 cost) as shown in Table 3.1.

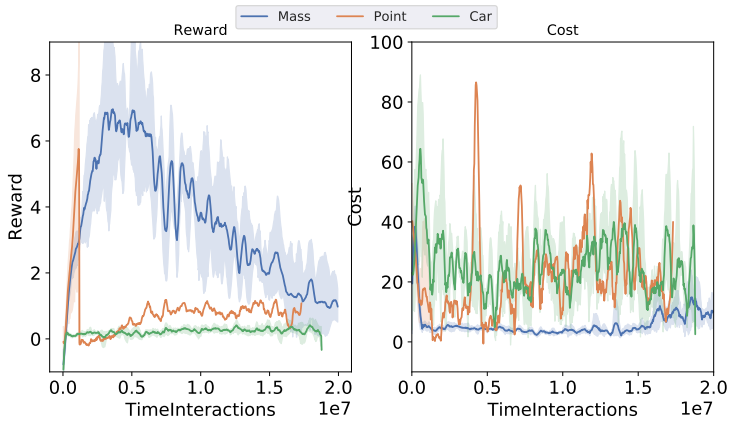
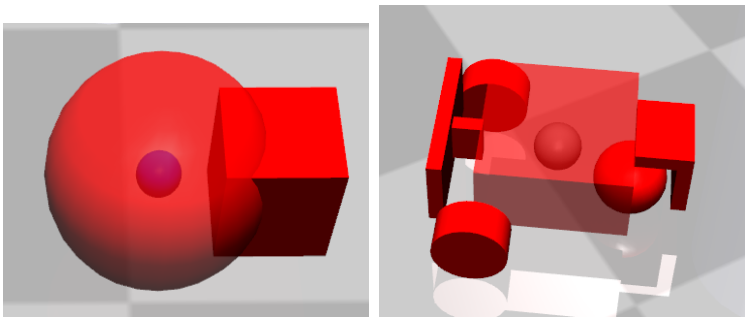


Figure B.3: Training curves of the goal-reaching policies used in "SAC + PPO Lag".



(a) Point (b) Car

Figure B.4: Root node position of different robots shown by the location of the small blue sphere.

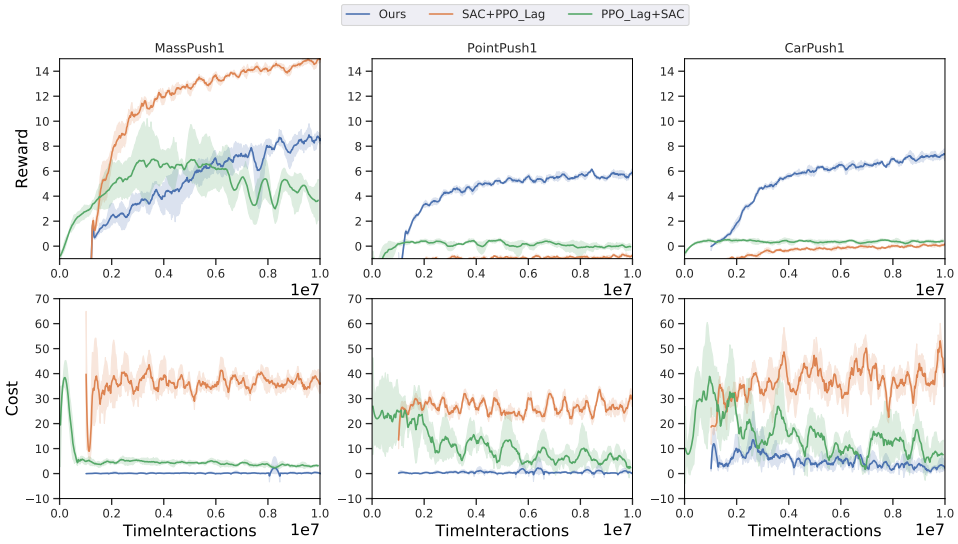


Figure B.5: Training curves of our method and 2 ablations (see Section 3.6.3 for details on these methods).

B.2 Implementation Details

B.2.1 Definition of "root" node

In MuJoCo, a robot is defined in a tree structure. Specifically, a robot is usually defined by mounting the adjacent link onto the previous link. The root node of the robot is the root body of the tree structure. In our experiments, it is usually the body of the robot. Thus the location of the root node can be interpreted as the center of the robot. The position of the root node is shown in Fig. B.4. The small blue sphere inside the robot denotes the position of the root node.

B.2.2 Additional details about the trajectory-following module

In the trajectory-following module, the trajectory is represented as a set of waypoints. However, the goal-following agent only takes in a single goal as input. Thus, we use the following procedure to select a waypoint from the set for the goal-following agent:

1. The agent keeps track of the waypoint that is input into the goal-following

agent ("tracking point"). The tracking point is initialized as the first waypoint of the trajectory.

2. At each time step, the agent searches from the tracking point to the end of the trajectory until it finds a waypoint that is at least d_{min} away from the current robot root position. It sets the point as the tracking point for the current time step. It is possible that the tracking point is the same as the previous time step.

B.2.3 Additional details about the trajectory optimizer

In our experiments, instead of using three distinctive Lagrangian values for the three constraints mentioned in equation 3.3 respectively, we use a single Lagrangian value λ for all 3 constraints, to simplify our implementation.

To solve the dual problem mentioned in Equation 3.8, instead of using gradient descent to update λ, in practice, we follow a simpler procedure of increasing λ if the obstacle constraints are violated until the optimizer returns a feasible solution.

B.2.4 Implementation details of training the goal-reaching policy

For our method and the "PPO Lag + SAC" ablation, we use SAC to train the goal-reaching policy. In our method, we sample the goal from a range of d_{min} to $2d_{min}$. Specifically, d_{min} is set to be $0.2m$ in our experiment to match the average distance between waypoints in the trajectory. For the "PPO Lag + SAC" ablation, we sample the goal randomly from a $2m \times 2m$ area to match the distribution of subgoals output from the high-level PPO Lagrangian agent.

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017. 2.2.1, 3.1, 3.2.1, 3.6.1

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 1.1, 2.1, 2.2.2, 3.1

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 1.1, 3.2.1

[4] Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3):387–417, 1998. 2.2.1, 2.3.3

[5] Eitan Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999. 2.3.2, 3.1, 3.2.1, 3.3

[6] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018. 3.2.3

[7] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 3.1

[8] Yarden As, Ilnura Usmanova, Sebastian Curi, and Andreas Krause. Constrained policy optimization via bayesian world models. *arXiv preprint arXiv:2201.09802*, 2022. 3.1, 3.2.1

[9] Tim D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Robotics: Science and Systems*, volume 10, pages 1–10. Citeseer, 2014. 3

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 2.1, 2.2.2

[11] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30, 2017. 2.2.1

[12] Mohak Bhardwaj, Byron Boots, and Mustafa Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020. 3.2.3

[13] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017. 2.2.1

[14] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018. 2.2.1

[15] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019. 2.2.1

[16] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993. 2.1, 2.2.2

[17] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998. 3.2.2

[18] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR, 2017. 2.2.2

[19] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005. 2.2.1

[20] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017. 2.2.2

[21] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. 3.5.4

[22] Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In *International Conference on Machine Learning*, pages 2535–2544. PMLR, 2019. 2.1

[23] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco. *arXiv preprint arXiv:2212.00541*, 2022. 3.2.2

[24] Taylor A Howell, Simon Le Cleac'h, Kevin Tracy, and Zachary Manchester. Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints. *arXiv preprint arXiv:2205.09255*, 2022. 1.1, 3.2.2

[25] Taylor A Howell, Simon Le Cleac'h, Sumeet Singh, Pete Florence, Zachary Manchester, and Vikas Sindhwani. Trajectory optimization with optimization-based dynamics. *IEEE Robotics and Automation Letters*, 7(3):6750–6757, 2022. 1.1, 3.2.2

[26] Xuemin Hu, Long Chen, Bo Tang, Dongpu Cao, and Haibo He. Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. *Mechanical systems and signal processing*, 100:482–500, 2018. 2.1

[27] Ashish Kumar Jayant and Shalabh Bhatnagar. Model-based safe deep reinforcement learning via a constrained proximal policy optimization algorithm. *arXiv preprint arXiv:2210.07573*, 2022. 3.2.1

[28] Yuqian Jiang, Fangkai Yang, Shiqi Zhang, and Peter Stone. Integrating task-motion planning with reinforcement learning for robust decision making in mobile robots. *arXiv preprint arXiv:1811.08955*, 2018. 3.2.3

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2.5.1

[30] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021. 3.1

[31] Sven Koenig and Maxim Likhachev. Dˆ* lite. *Aaai/iaai*, 15:476–483, 2002. 3.2.2

[32] M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010. 2.1, 2.2.2

[33] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 3.1

[34] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*,

2016. 3.2.3

[35] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003. 3.2.2

[36] Changliu Liu and Masayoshi Tomizuka. Robot safe interaction system for intelligent industrial co-robots. *arXiv preprint arXiv:1808.03983*, 2018. 2.1

[37] Zuxin Liu, Zhepeng Cen, Vladislav Isenbaev, Wei Liu, Steven Wu, Bo Li, and Ding Zhao. Constrained variational policy optimization for safe reinforcement learning. In *International Conference on Machine Learning*, pages 13644–13668. PMLR, 2022. 3.2.1

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1.1, 2.1, 3.1

[39] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4316–4336, 2020. 2.1

[40] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018. 3

[41] Gaoyang Pang, Geng Yang, and Zhibo Pang. Review of robot skin: A potential enabler for safe collaboration, immersive teleoperation, and affective interaction of future collaborative robots. *IEEE Transactions on Medical Robotics and Bionics*, 2021. 2.1

[42] Theodore J Perkins and Andrew G Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3(Dec):803–832, 2002. 2.2.1

[43] Harry A Pierson and Michael S Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017. 3.1

[44] Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, et al. Theseus: A library for differentiable nonlinear optimization. *arXiv preprint arXiv:2207.09442*, 2022. 3.5.2

[45] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011. A.2.1

[46] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019. 1.1, 2.1, 2.5, 2.5.2, 3.1, 3.1, 3.2.1, 3.6, 3.6.1, 3.6.1, 3.6.2, B.1.3

[47] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964. 3.6.2

[48] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015. 3.2.1

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2.3.3, 3.2.1

[50] Harshit Sikchi, Wenxuan Zhou, and David Held. Lyapunov barrier policy optimization. *arXiv preprint arXiv:2103.09230*, 2021. 2.2.1, 3.2.1

[51] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 1.1, 2.1

[52] Jarrod M Snider et al. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009. A.2.1

[53] Pengwei Wang, Song Gao, Liang Li, Binbin Sun, and Shuo Cheng. Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm. *Energies*, 12(12):2342, 2019. 2.1

[54] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999. 3.5.2

[55] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4583–4590. IEEE, 2021. 3.2.3

[56] Long Yang, Jiaming Ji, Juntao Dai, Linrui Zhang, Binbin Zhou, Pengfei Li, Yaodong Yang, and Gang Pan. Constrained update projection approach to safe policy optimization. *arXiv preprint arXiv:2209.07089*, 2022. 3.2.1

[57] Haonan Yu, Wei Xu, and Haichao Zhang. Towards safe reinforcement learning with a safety editor policy. *arXiv preprint arXiv:2201.12427*, 2022. 1.1, 3.2.1

[58] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014. 2.2.2