Thesis for Master's Degree

State-wise Safety in Autonomous Driving via Lagrangian-based Constrained Reinforcement Learning

Minseok Seo

Artificial Intelligence Graduate School

Gwangju Institute of Science and Technology

2025

석사학위논문

라그랑지안 기반 제약 강화학습을 이용한 자율주행에서의 상태별 안전성 고려

서민석

AI대학원(학과)

광주과학기술원

2025

State-wise Safety in Autonomous Driving via Lagrangian-based Constrained Reinforcement Learning

Advisor: Ue-Hwan Kim

by

Minseok Seo

Artificial Intelligence Graduate School

Gwangju Institute of Science and Technology

A thesis submitted to the faculty of the Gwangju Institute of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Artificial Intelligence Graduate School

Gwangju, Republic of Korea

June 20, 2025

Approved by

Professor Ue-Hwan Kim

Committee Chair

State-wise Safety in Autonomous Driving via Lagrangian-based Constrained Reinforcement Learning

Minseok Seo

Accepted in partial fulfillment of the requirements for

the degree of Master of Science

June 20, 2025

Committee Chair _____

Prof. Ue-Hwan Kim.

Committee Member _____

Prof. Kyung Joong Kim.

Committee Member _____

Prof. Kyunghwan Choi.

Dedicated to my family.

MS/AI 20231115 Minseok Seo. State-wise Safety in Autonomous Driving via Lagrangian-based Constrained Reinforcement Learning. Artificial Intelligence Graduate School. 2025. 49p. Advisor: Prof. Ue-Hwan Kim.

Abstract

For the practical deployment of autonomous driving systems, high levels of safety and adaptability are essential. Accordingly, Deep Reinforcement Learning (DRL), which learns and improves driving strategies through trial and error, has gained attention. However, the reward-driven nature of reinforcement learning may still lead to unsafe or abnormal behavior even after training. To address this limitation, Constrained Reinforcement Learning (CRL) has been proposed to balance safety and performance. While CRL typically defines constraints as expected cumulative costs, this formulation does not consider whether constraints are satisfied at each state, making it difficult to ensure state-wise safety. In this paper, we extend a Lagrangian-based CRL approach by estimating state-wise Lagrangian multipliers, allowing the policy to account for state-level safety. We evaluate the proposed method in OpenAI's Safety Gym environment and compare its performance with existing Lagrangian-based methods.

MS/AI 서민석. 라그랑지안 기반 제약 강화학습을 이용한 자율주행에서의 상태별
20231115 안전성 고려. AI대학원(학과). 2025. 49p. 지도교수: 김의환.

국 문 요 약

자율주행 시스템의 실제 적용을 위해서는 높은 안정성과 적응성이 요구된다. 이에 따라, 시행착오를 통해 주행 전략을 학습하며 발전시키는 심층 강화 학습(Deep Reinforcement Learning, DRL)이 주목받고 있다. 하지만 강화 학습은 본질적으로 보상을 극대화하는 방향으로 정책을 학습하기 때문에, 학습 후에도 안전하지 않거나 비정상적인 행동을 할 가능성을 완전히 배제하기 어렵다. 이러한 한계를 해결하기 위해, 정책 학습 시 안정성과 성능 간의 균형을 도모하는 제약 강화 학습(Constrained Reinforcement Learning, CRL)이 제안되었다. 제약 강화 학습은 기댓값 기반 누적 비용 형태의 제약 조건을 만족하도록 정책을 학습하지만, 각 상태에서의 제약 조건 충족 여부를 고려하지 않아 상태별 안정성을 보장하기 어렵다. 본 논문에서는 제약 강화 학습의 한 방식인 라그랑지안 기반의 방법을 확장하여, 상태별 라그랑주 승수를 추정함으로써 정책이 상태별 안정성을 고려하도록 한다. 또한 제안한 방법을 OpenAI의 시뮬레이션 환경인 Safety Gym을 통해 기존 라그랑지안 기반의 방법들과 비교하여 검증하였다.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Reinforcement Learning (RL) [1] is a method for learning an optimal policy through trial and error. Although its theoretical foundations have been established for several decades, its practical applications were limited by various challenges. One of the biggest challenges in reinforcement learning is extending it to continuous spaces, which leads to an increase in the dimensionality of state and action spaces. Due to the exponential growth in the number of possible states and actions, the corresponding rise in computational complexity poses a significant obstacle to learning in high-dimensional environments. To address this issue, traditional approaches often relied on handcrafted feature engineering to simplify the problem space. However, designing effective features by hand is both time-consuming and domain-specific, limiting the generalizability of learned policies across different tasks. The emergence of deep learning addressed this issue by enabling automatic feature extraction from raw, high-dimensional inputs such as images, sensor data. This advancement eliminated the need for manual feature design and allowed reinforcement learning agents to operate directly on raw observations. However, applying deep learning to reinforcement learning introduced another significant challenge: the data collected by agents is highly correlated. Unlike supervised

learning, where training data is typically assumed to be independent and identically distributed (IID), RL agents interact sequentially with the environment, resulting in temporally correlated data. This violates the IID assumption and can lead to instability and inefficient learning when training neural networks. A major breakthrough in overcoming these limitations came with the introduction of Deep Q-Network (DQN) [2,3] by DeepMind. By combining deep neural networks with Q-learning, DQN enabled agents to approximate complex value functions from high-dimensional inputs such as raw pixel images. This advancement allowed RL agent could achieve human-level performance in a variety of Atari games without relying on handcrafted features. This success of DQN has led to significant advances in the field of deep reinforcement learning (DRL), such as AlphaGo [4] and AlphaZero [5] by DeepMind, which demonstrated superhuman performance in board games like Go, Chess, and Shogi. In addition, OpenAI Five [6] showcased the power of DRL in complex, multi-agent environments by defeating professional human players in the real-time strategy game Dota 2. Another notable example is Dactyl [7], a robotic hand developed by OpenAI that learned to manipulate physical objects using reinforcement learning trained in simulation and successfully transferred to the real world, highlighting progress in sim-to-real transfer for robotic control. Despite these impressive achievements, applying reinforcement learning to real-world environments remains challenging. RL agents typically require a large number of iterations to learn effective policies, often relying on extensive exploration to discover rewarding behaviors. However, during this exploration process, agents may take unsafe or risky actions that can lead to catastrophic failures—particularly in safety-critical domains

such as robotics, autonomous driving, or healthcare. Moreover, even after training is complete, there is no guarantee that the learned policy will consistently behave safely, especially in unseen or out-of-distribution (OOD) environments. In particular, transferring policies from simulation to the real world (i.e., the sim-to-real problem) can cause even greater safety concerns when learned behaviors don't generalize well to the real world. A key underlying difficulty is the inherent challenge of designing reward functions that reliably induce safe and desirable behaviors across a wide range of situations.

1.2 Research Objective

In this thesis, we investigate how to learn safe policies in reinforcement learning through constrained optimization techniques, focusing on State-wise Constrained Reinforcement Learning (SCRL) [8], which introduces cost functions to enforce state-wise safety constraints during the learning process. Among various SCRL approaches, we examine Lagrangian-based methods due to their theoretical simplicity and empirical popularity. This thesis analyzes the limitations of existing Lagrangian methods in the SCRL setting and empirically examines how specific design choices, including the final-layer bias initialization and the learning rate of the Lagrange multiplier network, influence both performance and safety. We also propose a method, PPO Lagrangian Network, which extends Proximal Policy Optimization to the state-wise constraint setting using a Lagrange multiplier network. The proposed method is empirically evaluated against existing approaches on a range of tasks from the OpenAI Safety Gym.

1.3 Outline of the Thesis

This thesis is organized as follows:

- Chapter 2 provides background on reinforcement learning, including policy gradient methods, constrained reinforcement learning and state-wise constrained reinforcement learning. It also reviews prior work relevant to this thesis.

- Chapter 3 introduces the proposed method, PPO Lagrangian Network, which incorporates a state-wise Lagrange multiplier network into the PPO framework. The design and training procedure are detailed, along with comparisons to related methods.

- Chapter 4 presents experimental results. We first investigate the influence of key hyperparameters, such as the entropy coefficient α, bias initialization, and learning rate of the Lagrange multiplier network. We then evaluate the performance of PPO Lagrangian network across Safety Gym tasks.

- Chapter 5 concludes the thesis with a summary of findings and discusses limitations and directions for future research.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a framework in which an agent interacts with an environment and learns a policy to maximize cumulative rewards. The agent observes the state of the environment, takes actions, and receives rewards based on those actions. This process is formalized as a Markov Decision Process (MDP) [9], which provides a formal strcuture for modeling decision-making problems. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, P is the state transition probability function, R is the reward function, and $\gamma \in [0, 1)$ is the discount factor. In this thesis, we consider a finite-horizon setting and use the undiscounted return. The objective of RL is to find an optimal policy π^* that maximizes the expected cumulative reward, defined as:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} J(\theta) \\ J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r_t \right] \end{aligned} \tag{2.1}$$

The policy π_θ is assumed to be a differentiable function parmeterized by θ, denoted as $\pi_\theta(a|s)$, which represents the probability of taking ation a given state s. The expectation $\mathbb{E}_{\tau \sim \pi_\theta}$ is taken over the trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$ generated by

following the policy π_θ.

2.1.1 Policy Gradient Methods

Since the policy is differentiable, its gradient can be expressed using the likelihood ratio trick:

$$\begin{aligned} \nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \end{aligned} \tag{2.2}$$

This term $\nabla_\theta \log \pi_\theta(s,a)$ is referred to as the score function. Although we previously defined the objective function $J(\pi_\theta)$ as the expected cumulative reward, we now consider a simplified case to facilitate explanation. A one-step MDP, in which the agent takes an action from the initial state, receives a reward, and the episode terminates immediately. Then, the objective function can be written as (d is the initial state distribution):

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) R_{s,a} \end{aligned} \tag{2.3}$$

The gradient of the objective function can be computed as follows:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(s,a) R_{s,a} \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R_{s,a} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r] \end{aligned} \tag{2.4}$$

The policy gradient theorem generalizes this result to multi-step MDP, where the objective function is defined as the expected cumulative reward over multiple time steps. In other words, it replaces the instantaneous reward r with the long-term value $Q^{\pi_\theta}(s, a)$, the action value function.

Theorem 2.1.1 (Policy Gradient Theorem)**.**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)] \tag{2.5}$$

REINFORCE

In practice, the exact action value function $Q^{\pi_\theta}(s, a)$ is typically unknown. Accordingly, the estimated return G_t can be used as an approximation of the action value function. In other words, the action value function $Q^{\pi_\theta}(s, a)$ can be replaced with the return G_t from real sample trajectories using the Monte Carlo method.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) G_t] \tag{2.6}$$

This leads to the Monte-Carlo policy gradient method, commonly known as REINFORCE. However, REINFORCE suffers from high variance in the gradient estimates, due to its reliance on a full trajectory.

Actor-Critic

A common approach to reducing the variance is to use a critic approximates the action value function, $Q_\phi(s,a) \approx Q^{\pi_\theta}(s,a)$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\phi(s,a)] \tag{2.7}$$

This is the basic idea of the Actor-Critic methods. Actor updates the policy parameters θ and a critic updates the value function parameters w.

Advantage Actor-Critic

To further reduce the variance, we can introduce a baseline function $B(s)$. Importantly, subtracting a baseline from the action value function does not change the gradient because its gradient is zero.

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) B(s)] &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) B(s) \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(s,a) B(s) \\
&= \sum_{s\in\mathcal{S}} d(s) B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \\
&= 0
\end{aligned} \tag{2.8}$$

Therefore, subtracting a baseline function from the action value function not only leaves the gradient unchanged but also reduces its variance.

$$\begin{aligned} A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)] \end{aligned} \tag{2.9}$$

However, since the exact advantage function is generally inaccessible, both the action value function $Q^{\pi_\theta}(s,a)$ and the value function $V^{\pi_\theta}(s)$ must be approximated. One common approach is to use the temporal difference (TD) error δ^{π_θ}, which is an unbiased estimator of the advantage function. To summarize, the policy gradient has many equivalent formulations:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) v_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\phi(s,a)] && \text{Q Actor-Critic} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)] && \text{Advantage Actor-Critic} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta^{\pi_\theta}] && \text{TD Actor-Critic} \end{aligned} \tag{2.10}$$

Proximal Policy Optimization (PPO)

The methods introduced above are all on-policy: training samples are collected using the same policy that we want to optimize. However, on-policy methods can be inefficient in terms of sample usage and may suffer from instability due to large updates. Proximal Policy Optimization (PPO) [10] addresses these issue by incorporating importance sampling to reuse data collected from the old policy and introducing a clipped surrogate

objective function to prevent large policy updates that may cause divergence. Based on the Advantage Actor-Critic formulation Eq. 2.9, the policy gradient can be interpreted as optimizing the following objective function:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\log \pi_\theta(s,a) A^{\pi_\theta}(s,a)] \tag{2.11}$$

This formulation assumes that data is collected from the current policy. However, in order to improve sample efficiency by reusing data from a previous policy $\pi_{\theta_{\text{old}}}$, we can apply importance sampling. So, the objective function can be rewritten as:

$$J^{\text{TRPO}}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(s,a)}{\pi_{\theta_{\text{old}}}(s,a)} A^{\pi_{\theta_{\text{old}}}}(s,a) \right] \tag{2.12}$$

This objective was introduced in Trust Region Policy Optimization (TRPO) [11], the predecessor of PPO. In this formulation, $r(\theta) = \frac{\pi_\theta(s,a)}{\pi_{\theta_{\text{old}}}(s,a)}$ represents the probability ratio between old and new policies. While, this allows us to reuse data from the old policy, maximizing this objective without any constraint can lead to large updates and unstable training. To prevent such instability, PPO imposes the constraint by forcing $r(\theta)$ to stay within a small interval $[1-\epsilon, 1+\epsilon]$.

$$J^{\text{PPO}}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\min \left(r(\theta) A^{\pi_{\theta_{\text{old}}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_{\text{old}}}}(s,a) \right) \right] \tag{2.13}$$

The function $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the probability ratio to the range $[1-\epsilon, 1+\epsilon]$. Therefore, the objective function takes the minimum one between the original value

and the clipped value. This ensures that the policy update does not deviate too far from the old policy, thus stabilizing the training process.

2.1.2 Off-Policy Gradient Methods

Off-policy methods allow the agent to learn from data collected by a different policy called the behavior policy. As a result, the off-policy approach improves sample efficiency by enabling the reuse of past experiences and enhances exploration by allowing the agent to learn from data collected by a behavior policy. In order to apply off-policy learning within the policy gradient framework, we can incorporate importance sampling into the policy gradient Theorem 2.1.1.

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)] \\ &= \mathbb{E}_{\tau\sim\beta}\left[\frac{\pi_\theta(s,a)}{\beta(s,a)} \nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right] \end{aligned} \tag{2.14}$$

This allows to estimate the gradient using samples generated by a behavior policy β instead of the current policy π_θ.

Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [12–14] is an off-policy actor-critic model. In SAC, the policy is trained with the objective of maximizing the expected return and the entropy of the policy.

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}(r_t + \alpha\mathcal{H}(\pi_\theta(\cdot|s_t)))\right] \tag{2.15}$$

$$\mathcal{H}(\pi_\theta(\cdot|s_t)) = \mathbb{E}_{a_t\sim\pi_\theta}[-\log \pi_\theta(a_t|s_t)] \tag{2.16}$$

where $\mathcal{H}(\cdot)$ is the entropy measure and α controls how important the entropy term is relative to the reward. Entropy maximization leads to policies that can explore more and capture multiple modes of near-optimal strategies. If there exist multiple options that seem equally good, the policy should assign each of them an equal probability. In the SAC algorithm, the policy is updated through the soft policy iteration. First, the policy evaluation step of soft policy iteration computes the soft Q-value:

$$Q^{\pi_\theta}(s,a) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\left(r_t + \alpha\mathcal{H}(\pi_\theta(\cdot|s_t))\right)\middle| s_0 = s, a_0 = a\right] \tag{2.17}$$

Then, the policy improvement step updates the policy to minimize the KL-divergence:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{\mathrm{KL}}\left(\pi_\theta(\cdot|s_t)\middle\|\frac{\exp(Q_\theta(s_t,\cdot))}{z_\theta(s_t)}\right)\right] \\ &= \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\theta}[\alpha\log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t,a_t)]\right] \end{aligned} \tag{2.18}$$

2.2 Constrained Reinforcement Learning

Constrained Reinforcement Learning (CRL) extends the standard RL framework by incorporating constraints into the learning process. CRL is formalized as a Constrained Markov Decision Process (CMDP) [15], which is defined by a tuple $\langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\mathcal{C},\gamma\rangle$. Here, $\mathcal{C}$ is the cost functions associated with the constraints. The feasible policy set in a CMDP is given by:

$$\Pi_C = \{\pi : J_{c_i}(\pi) \le d_i, \quad i = 1,\cdots,k\} \tag{2.19}$$

where $J_{c_i}(\pi)$ is a cost-based constraint function defined the expected cumulative cost, and d_i is the threshold for the i-th constraint. The objective of CRL is to find an optimal policy that maximizes the expected cumulative reward while satisfying the constraints. In the context of policy gradient methods, the constrained optimization problem can be formulated as follows:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} J(\theta) \\ J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} r_t\right] \text{ subject to } \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} c_t\right] \leq d \end{aligned} \tag{2.20}$$

2.2.1 Lagrangian Method

Constrained optimization problem defined in Eq. 2.20 can be solved using various methods. In this thesis, however, we consider only the Lagrangian method. By applying Lagrangian relaxation, the constrained optimization problem can be transformed into an unconstrained optimization problem, in which the constraint is incorporated into the objective function using a Lagrange multiplier λ.

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \mathcal{L}(\theta, \lambda) \\ \mathcal{L}(\theta, \lambda) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} r_t\right] - \lambda\left(\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} c_t\right] - d\right) \end{aligned} \tag{2.21}$$

The Lagrange multiplier λ is a non-negative scalar that adjusts the trade-off between maximizing the expected cumulative reward and satisfying the constraints.

2.2.2 Related Work: PPO Lagrangian

PPO Lagrangian is an adaptation of PPO to the CRL setting by incorporating the Lagrangian method [16]. Thus, the objective function of PPO Lagrangian can be written as:

$$J^{\text{PPO-Lag}}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[\min\left(r(\theta)A^{\pi_{\theta_{\text{old}}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(s,a)\right) - \lambda r(\theta)A_c^{\pi_{\theta_{\text{old}}}}\right] \tag{2.22}$$

where λ is the Lagrange multiplier that adjusts the trade-off between the reward and the cost and $A_c^{\pi_{\theta_{\text{old}}}}$ is the advantage function for the cost. The Lagrange multiplier λ is updated iteratively during training to ensure the constraint is satisfied. The update of the Lagrange multiplier depends on whether the empirical cumulative cost exceeds the threshold d. Specifically, the empirical cumulative cost is computed as:

$$\hat{J}_c = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} c_t^{(i)} \tag{2.23}$$

where N is the number of sampled episodes, and $c_t^{(i)}$ denotes the cost received at time step t in the i-th episode. The Lagrange multiplier λ is updated via gradient ascent:

$$\lambda \leftarrow \left[\lambda + \beta\left(\hat{J}_c - d\right)\right]_+ \tag{2.24}$$

where β is the leraning rate, and $[\cdot]_+$ denotes clipped to be non-negative. This update increases λ when the empirical cost exceeds the threshold d, encouraging the policy to reduce the cost in subsequent updates. The overall structure of PPO Lagrangian is

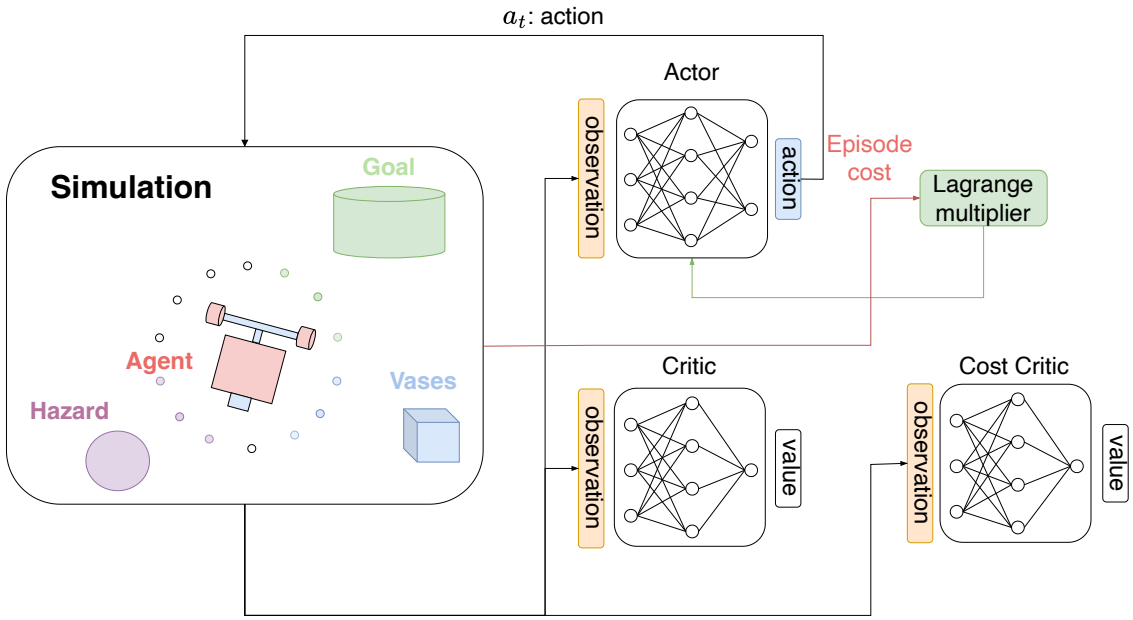


Figure 2.1: Structure of PPO Lagrangian

illustrated in Fig. 2.1. The agent interacts with the environment by taking actions generated from the actor network, thereby collecting trajectories. Using the actual rewards and costs obtained from the trajectories, the value functions are updated, and both reward and cost advantages are computed. The Lagrange multiplier is updated based on the cost returns from the trajectories. Finally, the policy is updated using the reward advantage, cost advantage, and the Lagrange multiplier according to Equation 2.22.

2.3 State-wise Constrained Reinforcement Learning

State-wise Constrained Reinforcement Learning (SCRL) is a variant of CRL that imposes constraints at the state level. CRL considers the cumulative cost over the entire trajectory, while SCRL focuses on the cost at each transition. SCRL is formalized as a State-wise Constrained Markov Decision Process (SCMPD), it is quite similar to

CMDP, but SCMDP enforces the constraint for every state action trasition satisfies a hard constraints. The objective of SCRL is to find an optimal policy that maximizes the expected cumulative reward while satisfying the state-wise constraints.

$$\begin{aligned} \pi^* &= \arg\max_{\pi_\theta} J(\theta) \\ J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} r_t\right] \text{ subject to } \mathbb{E}_{\tau\sim\pi_\theta}[c(s,a)] \leq w, \quad \forall s \in S \end{aligned} \tag{2.25}$$

2.3.1 Related Work: Feasible Actor-Critic

Feasible Actor-Critic (FAC) is an extension of the Soft Actor-Critic algorithm to the SCRL setting [17]. In FAC, the state-wise constraint is enforced via a cost action-value function, formulated as:

$$Q_c^{\pi_\theta}(s,a) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} c_t | s_0 = s, a_0 = \ a\right] \leq w \tag{2.26}$$

The objective function of FAC is defined as:

$$J^{\text{FAC}}(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\theta}[\alpha\log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t,a_t) + \lambda_\xi(s_t)\left(Q_{\phi_c}(s_t,a_t) - w\right)]\right] \tag{2.27}$$

where Q_{ϕ_c} is the cost action-value function, and λ_ξ is the Lagrange multiplier network, which estimate the Lagrange multiplier for each state. The update of the Lagrange multiplier network depends on whether the cost action-value function exceeds the threshold w.

$$J_\lambda(\xi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\theta}\left[\lambda_\xi(s_t)\left(Q_{\phi_c}(s_t,a_t) - w\right)\right]\right] \tag{2.28}$$

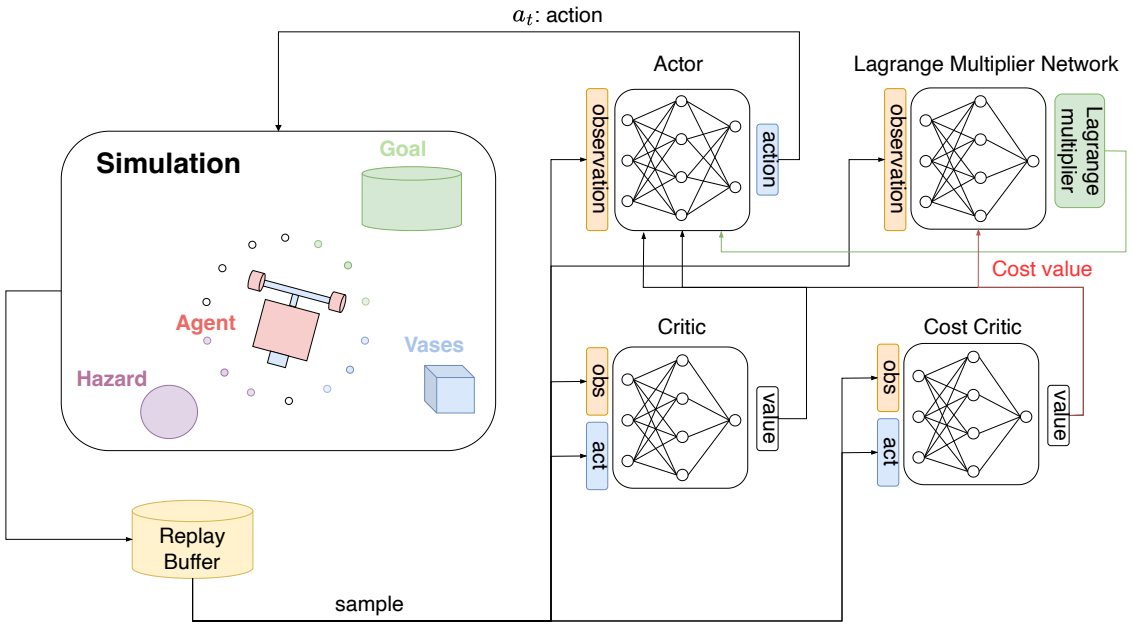


Figure 2.2: Structure of Feasible Actor-Critic

Fig. 2.2 illustrates the architecture of Feasible Actor-Critic (FAC). Since FAC is an off-policy algorithm, samples collected during the rollout phase are stored in a replay buffer, and mini-batches are drawn from it to update the actor, critics, and Lagrange multiplier network. The reward and cost critic networks are used to update their respective Q-functions. Using the reward and cost values estimated by the critic networks, together with the Lagrange multiplier computed from the Lagrange multiplier network, the policy is updated according to Equation 2.27. The Lagrange multiplier network is updated based on the cost values estimated by the cost Q-function. Although FAC contributes by proposing a framework that leverages a Lagrange multiplier network to address state-wise safety in policy learning, it has several limitations.

- Soft Actor-Critic (SAC) encourges exploration and promotes diverse action selection by adjusting the temperature parameter α. However, this objective can

conflict with the constraint penalty term, which pushes the policy toward satisfying constraints. As a result, it becomes more difficult for the policy consistently satisfy the constraints.

- Instead of using the empirical cost values, FAC relies on a cost value estimated by the cost action-value function $Q_{\phi_c}(s, a)$. This introduces instability in the update of the Lagrange multiplier network due to potentially inaccurate cost estimates, which in turn can lead to unreliable policy updates.

Chapter 3

PPO-based Method in State-wise Constrained RL

3.1 PPO Lagrangian Network

As discussed in Section 2.3.1, Feasible Actor-Critic (FAC) exhibits potential limitations arising from the characteristics of the SAC algorithm, its off-policy nature, and the way state-wise safety is defined through the cost Q-function. In this chapter, we introduce our proposed method, which extends PPO under the state-wise constrained RL framework. Similar to FAC method, we employ a network that estimates state-wise Lagrange multipliers to enforce safety constraints at the state level. Thus, objective function of the policy is modified to include a Lagrange multiplier network:

$$J^{\text{PPO-Lagnet}}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}}\Big[\min\big(r(\theta)A^{\pi_{\theta_{\text{old}}}}(s,a)\text{clip}(r(\theta),1-\epsilon,1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(s,a)\big) - \lambda_\xi(s)r(\theta)A_c^{\pi_{\theta_{\text{old}}}}\Big] \tag{3.1}$$

where $A_c^{\pi_{\theta_{\text{old}}}}$ is the advantage function for the cost and $\lambda_\xi(s)$ is the Lagrange multiplier network that estimates the Lagrange multiplier for the cost at state s. The Lagrange multiplier network is iteratively updated during training to guide the policy toward satisfying the imposed constraints. The update of the Lagrange multiplier network

depends on whether the empirical cost at state s exceeds the threshold w:

$$\lambda(s) \leftarrow \lambda(s) + \beta(\hat{J}_c - w) \tag{3.2}$$

The overall architecture of the PPO Lagrangian Network is illustrated in Fig. 3.1. The structure closely follows that of PPO Lagrangian described in Fig. 2.1: the agent interacts with the environment using actions generated by the actor network, and the collected trajectories are used to compute reward and cost advantages based on value function estimations. These quantities are then used to update the policy according to Equation 3.1. The key distinction from the PPO Lagrangian lies in the design of the Lagrange multiplier. Instead of using a single scalar multiplier, the PPO Lagrangian Network introduces a dedicated neural network that estimates a state-wise Lagrange multiplier $\lambda_\xi(s)$. To ensure the positivity of the Lagrange multiplier, a softplus activation function is applied to the final layer of the network. Moreover, this Lagrange multiplier network is updated based on the cost incurred at each individual state within the trajectories, allowing the algorithm to enforce constraints more precisely at the state level rather than relying on trajectory-level cost.

3.1.1 Comparison with PPO Lagrangian

In PPO Lagrangian, the Lagrange multiplier is a scalar because it enforces that the cumulative cost along trajectories generated by the policy π_θ remains below a specified threshold. In contrast, PPO Lagrangian Network estimates a state-wise Lagrange multiplier, which enables the policy to satisfy cost constraints more precisely at

Algorithm 1: PPO Lagrangian Network

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0, initial cost value function parameters ϕ_0^c, Lagrange multiplier network parameters ξ, threshold w

for *each epoch* $k = 0, 1, 2, \cdots$ **do**

  for *each time step* $t = 1$ **to** T **do**

    `// Collect trajectories`

    Sample action $a_t \sim \pi_{\theta_{t-1}}(s_t)$

    Execute action a_t in the environment and observe reward r_t, cost c_t and next state s_{t+1}

    Store transition $\tau_t = (s_t, a_t, r_t, c_t, s_{t+1})$ in buffer B_k

    if *episode end* **then**

      Compute rewards-to-go $\hat{R}_t$ and advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k}

      Compute costs-to-go $\hat{R}_t^c$ and cost advantage estimates $\hat{A}_t^c$ based on the current cost value function $V_{\phi_k}^c$

  `// Lagrange multiplier network update`

  Update the Lagrange multiplier λ by gradient ascent:

$$\xi_{k+1} = \xi_k + \beta \left[\frac{1}{|B_k|T} \sum_{\tau \in B_k} \sum_{t=0}^{T} (c_t - w) \right]$$

  `// Policy update`

  Update the policy parameters θ by maximizing the objective function:

$$\theta_{k+1} = \arg\max \frac{1}{|B_k|T} \sum_{\tau \in B_k} \sum_{t=0}^{T} \min \left(r_t(\theta) A^{\pi_{\theta_k}(s_t, a_t)},\ g\left(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right) \right) - \lambda_\xi(s_t) r_t(\theta) \hat{A}_t^{c, \pi_{\theta_k}(s_t, a_t)}$$

$$\text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)},\ g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

  `// Value function update`

  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|B_k|T} \sum_{\tau \in B_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

$$\phi_{k+1}^c = \arg\min_\phi \frac{1}{|B_k|T} \sum_{\tau \in B_k} \sum_{t=0}^{T} \left(V_{\phi^c}(s_t) - \hat{R}_t^c \right)^2$$

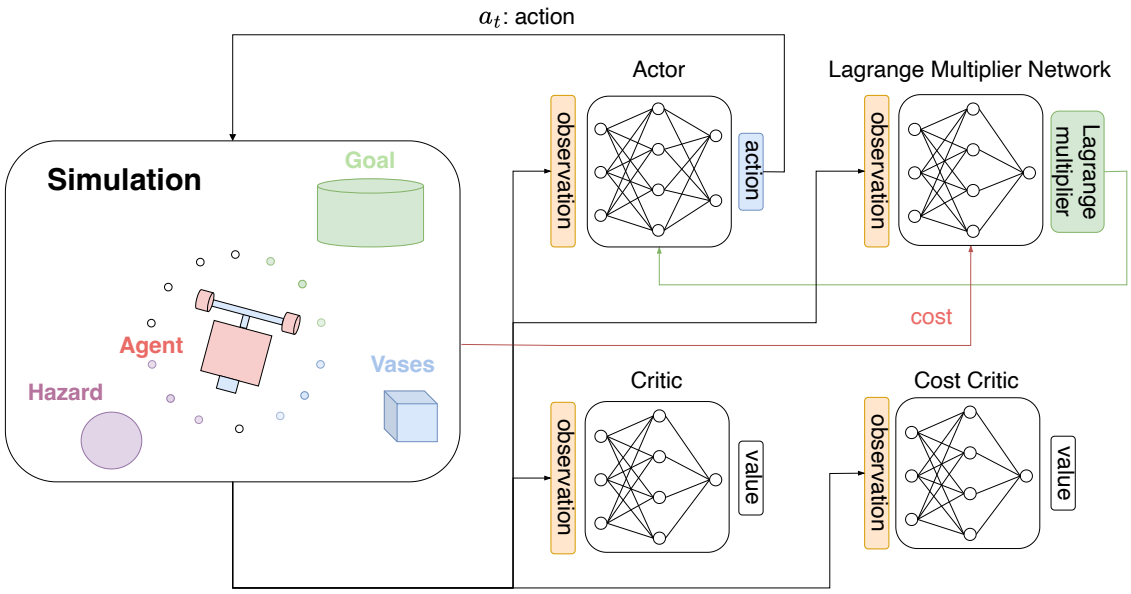


Figure 3.1: Structure of PPO Lagrangian Network

each individual state by enforcing the cost at each state to remain below the specified threshold.

3.1.2 Comparison with Feasible Actor-Critic

Feasible Actor-Critic (FAC) is similar to PPO Lagrangian Network in that it also estimates a state-wise Lagrange multiplier. However, unlike FAC, our method updates the Lagrange multiplier network in a different way. In FAC, the update is based on the output of the cost action-value network, which estimates the state-wise cost value. In contrast, our proposed method updates the policy based on the actual cost values observed for each state.

Chapter 4

Experiments

4.1 Setup

For our experiments, we use the OpenAI Safety Gym [16, 18] environments. OpenAI no longer maintains Safety Gym, and its development has been continued since 2023 by the PKU-Alignment Team under the name Safety Gymnasium. Safety Gymnasium preserves full compatibility with the original Safety Gym environments, including identical reward functions, cost functions, agents, and other components. To avoid confusion, we refer to these environments as Safety Gym throughout this thesis. In Safety Gym environements, there are three types of agents: Point, Car, and Doggo. In our experiments, we use the Car agent provided in Safety Gym.

- **Car**: A car-like robot with two independently controlled front wheels and a passive rear wheel. Since both forward movement and turning must be controlled together, it is slightly more difficult to operate than the Point. The Car agent is shown in Fig. 4.1.

Also, there are three types of tasks: Goal, Button, and Push. We use two of the tasks provided in Safety Gym: Goal and Button.

- **Goal**: The agent must reach a goal location while avoiding obstacles.
- **Button**: The agent must press a button while avoiding obstacles.

- **Push**: The agent must push yellow box to a goal location while avoiding obstacles.

The reward function provides positive rewards as the agent moves closer to the goal, and negative rewards as it moves farther away: $r_t = d_{t-1} - d_t$, where d_t is the distance to the goal at time t. Additionally, each task provides an extra positive reward when the agent successfully completes the objective, such as reaching the goal, pressing a button, or moving a box.

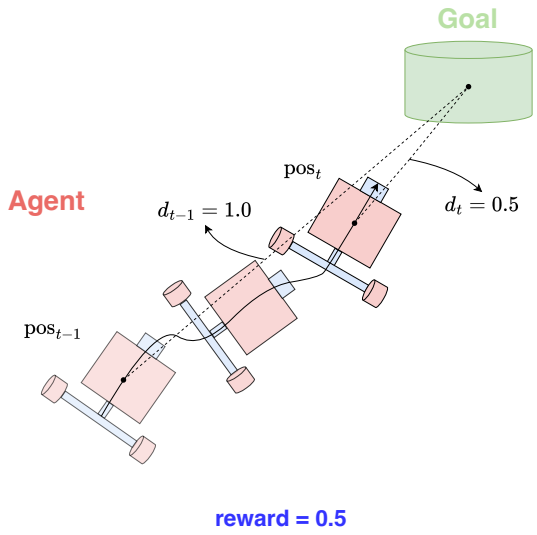


Figure 4.1: Illustration of how reward is computed in the Safety Gym

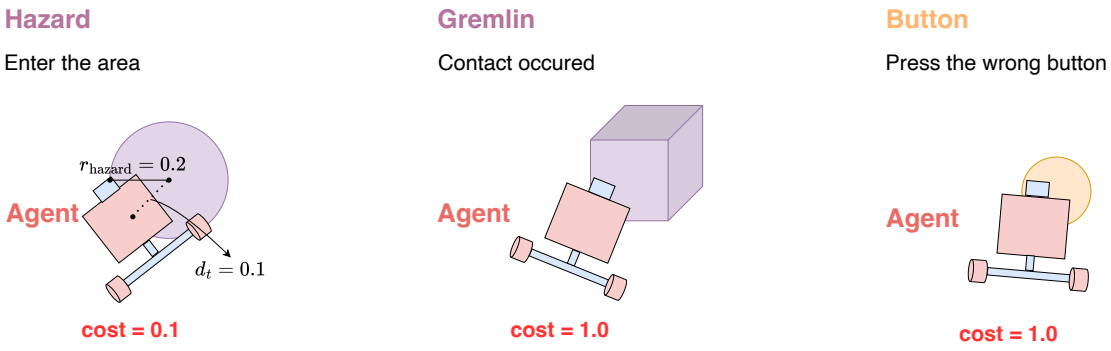


Figure 4.2: Illustration of how costs are computed in the Safety Gym

The cost function varies depending on the types of obstacles present in each task.

In our experimental environments, three types of obstacles are used, as illustrated in Fig. 4.2:

- **Hazard**: A cost is assigned when the agent enters a hazard area. The cost is computed as $c_t = d_t - r_{\text{hazard}}$, where d_t is the distance from the agent to the center of the hazard and $r_{\text{hazard}} = 0.2$ is the hazard radius.
- **Gremlin**: A cost of -1.0 is incurred when the agent makes contact with the gremlin object.
- **Button**: A cost of -1.0 is assigned when the agent presses a button that is not the designated goal button.

A key distinction of this thesis compared to other works lies in how the cost function is defined. Safety Gym provides two options for computing cost: a dense formulation and a sparse (indicator) cost. Many prior works adopt the sparse cost, where the environment returns 1 if a cost-triggering event occurs and 0 otherwise. In contrast, this thesis uses the dense cost described above, where the cost value is computed at each time step based on factors such as distance to obstacles or safety-related conditions. This choice is motivated by the fact that using sparse costs makes it difficult to precisely define and enforce constraint conditions. Dense costs allow for more precise and informative constraint definitions compared to sparse costs.

4.2 Analyzing the Influence of alpha in the Feasible Actor-Critic

As discussed in Section 2.1.2, Soft Actor-Critic (SAC) promotes stochastic exploration by maximizing both expected return and policy entropy. The temperature parameter α controls the importance of the entropy term: larger values of α encourage more exploratory and diverse action selection. However, in the Feasible Actor-Critic (FAC) framework, this exploration objective can conflict with the effect of the Lagrange multiplier, which encourages the policy to satisfy safety constraints. To examine this trade-off, we compare the training performance of unconstrained SAC (yellow curve) and FAC with various fixed α values, using Fig. 4.3, Fig. 4.4. In Figure 4.4, although small spikes are present, FAC with all tested α values generally maintains a lower cost compared to unconstrained SAC. This indicates that the constraint-satisfying mechanism in FAC is working as designed, leading to lower cost return by enforcing state-wise constraints more effectively. On the other hand, the return curves in Figure 4.3 reveal that with $\alpha = 0.001$ (green), the return improves rapidly in the early phase (up to around 200 epochs), but then gradually declines and converges toward zero. This behavior suggests that when α is relatively large, the entropy term dominates the learning signal, making the policy overly stochastic and less sensitive to both reward and constraint signals—ultimately degrading policy performance. Based on these observations, we fix α to a small value of 0.00001 for all subsequent comparisons between FAC and our proposed method, to minimize the entropy-induced instability while retaining some degree of exploration.

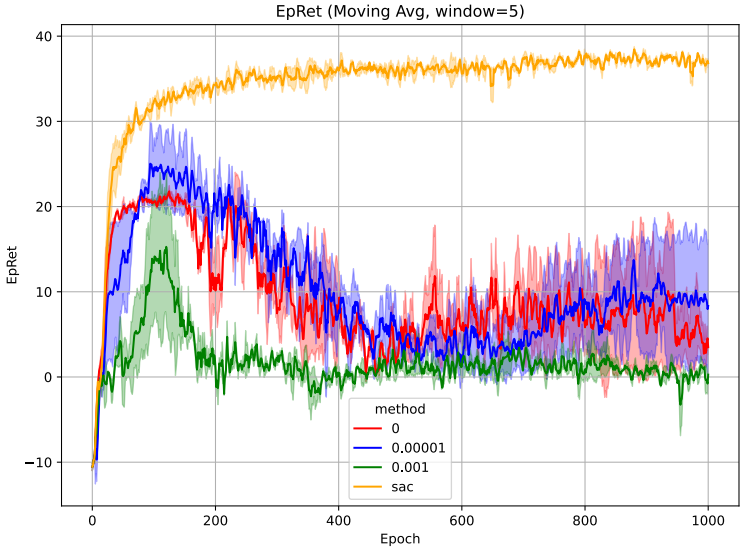


Figure 4.3: Training curves of return over epochs for different temperature parameters α in the Feasible Actor-Critic algorithm

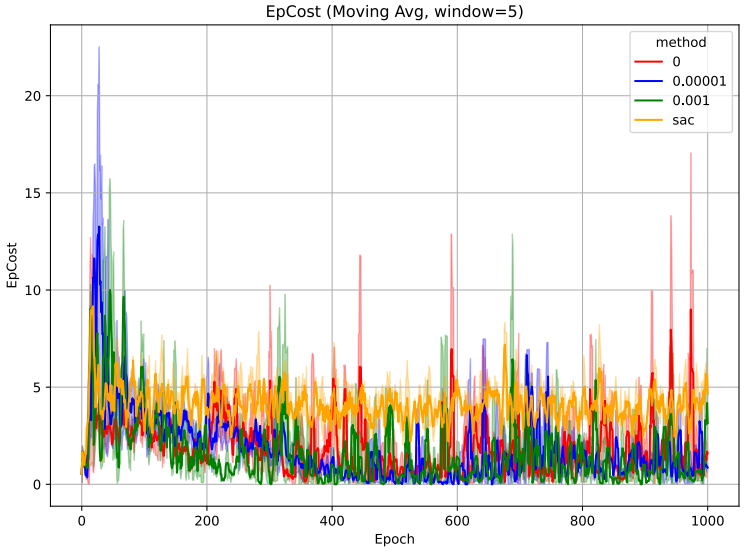


Figure 4.4: Training curves of cost return over epochs for different temperature parameters α in the Feasible Actor-Critic algorithm

Table 4.1: Hyperparameters used in Experiment 4.2

Hyperparameter	**Value**
Environment	Car Goal
Number of epochs	1000
Number of steps per epoch	2000
Batch size	256
Hidden layer size (MLP)	256
gamma (discount factor)	0.99
polyak (target network update)	0.995
Learning rate (policy)	5e-6
Learning rate (value function)	1e-3
Learning rate (Lagrange multiplier)	5e-8
Lagrange multiplier init (bias)	0

4.3 Analyzing the Influence of Bias Initialization in the Lagrange Multiplier Network of PPO Lagrangian Network

Lagrangian-based methods are known to be sensitive to both initialization and learning rate [19]. Improper initialization of the Lagrange multiplier network can lead to suboptimal or delayed policy convergence. In particular, a large initial value may overly penalize the policy, driving it prematurely toward a local optimum, while a small initial value may result in slow adaptation of the constraint enforcement. To analyze the effect of bias initialization, we conducted experiments by varying the initial bias value of the final layer in the Lagrange multiplier network. Specifically, we manually initialized the bias term of the last linear layer to one of four values: 0, 20, 40, and 60.

In Fig. 4.5, shows the training curves of return over 1000 epochs for different bias initializations of the Lagrange multiplier network. Although the convergence behavior varies across initial values, all four settings demonstrate stable policy learning overall.

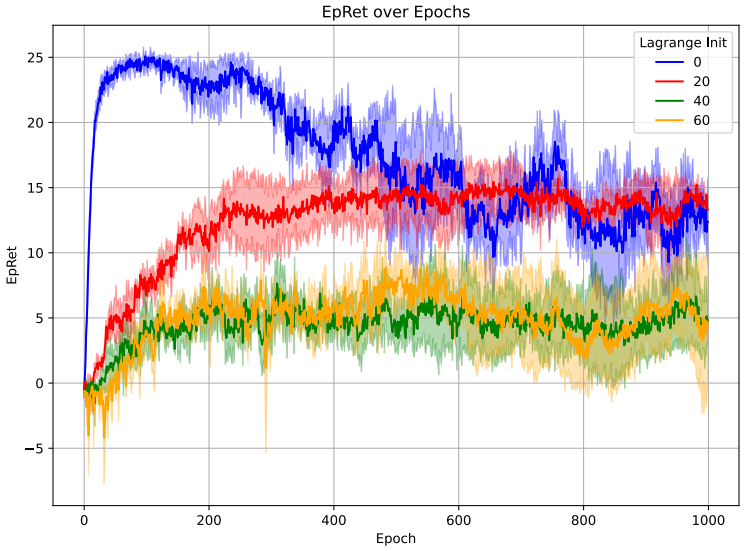


Figure 4.5: Training curves of return over epochs for different bias initializations in the Lagrange multiplier network

When the initial bias is set to 0, the early phase (epoch $0 \sim 300$) closely resembles unconstrained PPO, showing a rapid increase in return. However, as the Lagrange multiplier values grow, the return gradually decreases, indicating stronger constraint enforcement over time. With an initial bias of 20, the agent achieves the most stable and effective learning performance, maintaining high return while satisfying constraints. In contrast, higher initializations (40 and 60) result in excessive penalty signals, which overly constrain the policy. This leads to premature convergence to suboptimal solutions, likely due to limited exploration or overly conservative policy updates.

As shown in Fig. 4.6, all four settings demonstrate a stable reduction in cost over the course of training. This indicates that the Lagrange multiplier network, regardless of initialization, is able to learn to enforce constraints effectively. However, when comparing the results more closely, we observe that the setting with an initial bias of 0 achieves less cost reduction compared to the setting with an initial bias of 20,

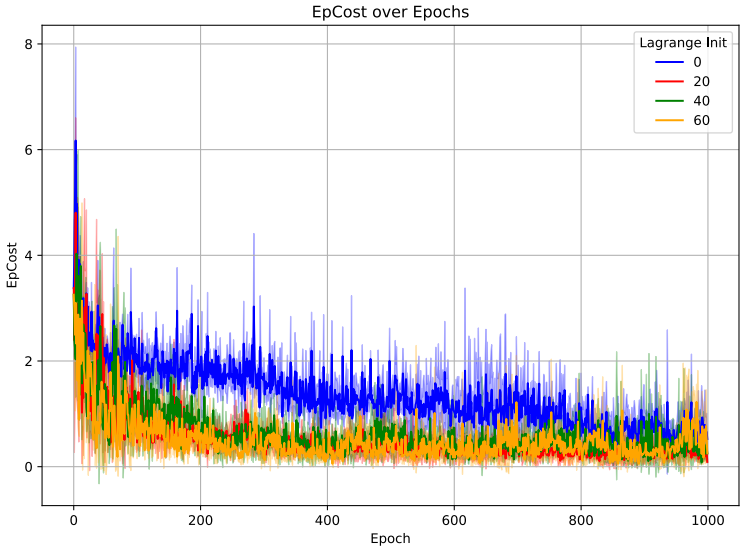


Figure 4.6: Training curves of cost return over epochs for different bias initializations in the Lagrange multiplier network

despite both eventually reaching similar Lagrange multiplier values (see Fig. 4.7). This suggests that early enforcement of constraints plays an important role in guiding the policy to safer regions of the state space. In the zero-initialization case, the Lagrange multiplier grows more slowly, allowing higher constraint violations in the early phase of training. As a result, the policy may have already been shaped in a way that is less sensitive to cost signals, making later enforcement of constraints less effective and leading to suboptimal cost minimization. Therefore, initializing the Lagrange multiplier with a moderately large value (e.g., 20) helps stabilize training by better balancing the trade-off between reward and cost throughout learning.

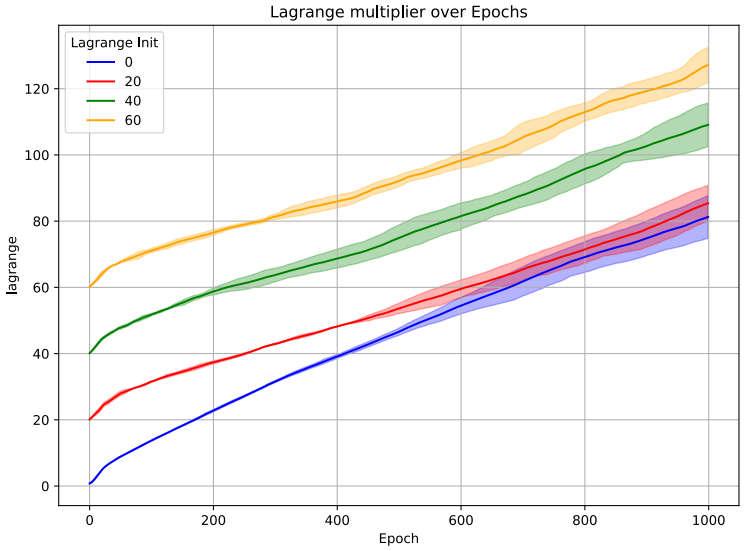


Figure 4.7: Training curves of Lagragne multiplier over epochs for different bias initializations in the Lagrange multiplier network

4.4 Analyzing the Influence of Learning Rate in the Lagrange Multiplier Network of PPO Lagrangian Network

The learning rate of the Lagrange multiplier network also plays a crucial role in stability and convergence. If the learning rate is too high, the multiplier may oscillate excessively, causing unstable policy updates. Conversely, if it's too low, the multiplier may adapt too slowly to apply appropriate penalties. To analyze the effect of the learning rate in the Lagrange multiplier network, we conducted experiments by varying the learning rate used for optimizing the network parameters. Specifically, we tested three different learning rate values: 0.0003, 0.003, and 0.03.

Fig 4.8 - 4.10 show the impact of learning rate on return, cost, and the Lagrange multiplier. Except for the case with a learning rate of 0.0003, all other settings converge to near-zero returns during training. In the case of a 0.003 learning rate, the return

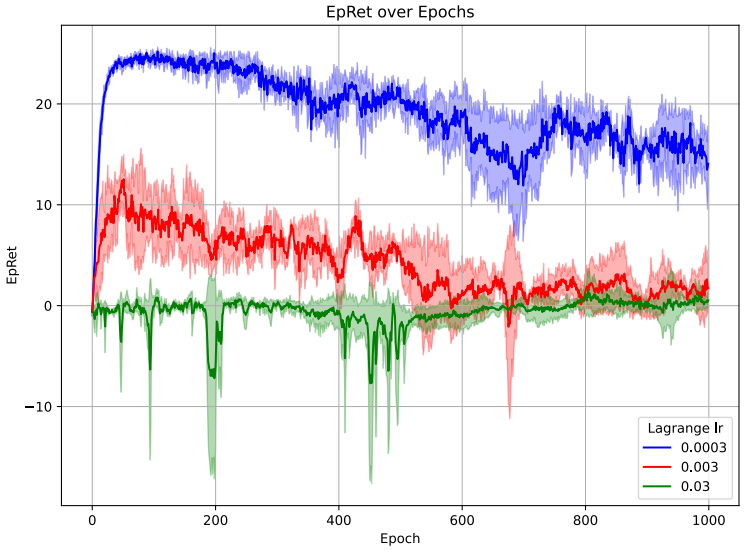


Figure 4.8: Training curves of return over epochs for different learning rate initializations in the Lagrange multiplier network

begins to decrease slightly after around epoch 400. During this period, the Lagrange multiplier continued its steady growth, increasing from around 200 at epoch 400 to approximately 500 by epoch 1000. This suggests that stronger constraint enforcement over time leads to a drop in reward. With a learning rate of 0.03, the Lagrange multiplier grows rapidly and exhibits high variance, making the overall training highly unstable. These findings suggest that the learning rate of the Lagrange multiplier network must be carefully tuned to ensure stable training while appropriately enforcing the constraints. Therefore, in the subsequent experiments, we initialize the bias of the Lagrange multiplier network to 20 and set the learning rate to 0.0003.

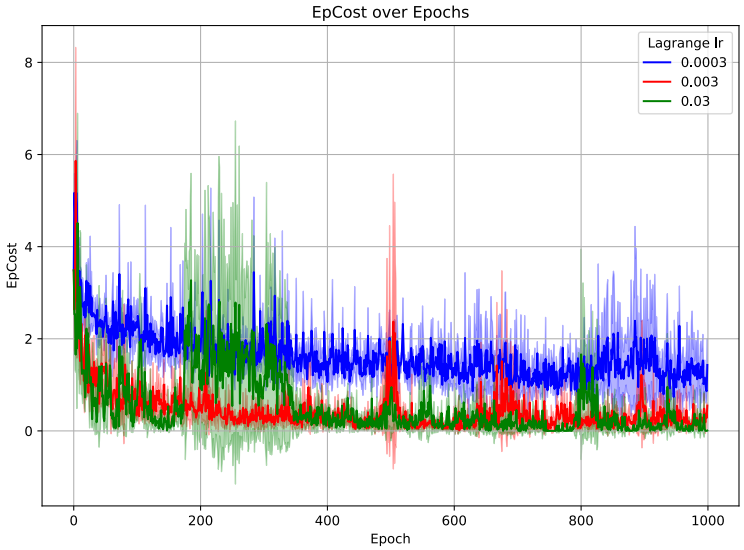


Figure 4.9: Training curves of cost return over epochs for different learning rate initializations in the Lagrange multiplier network

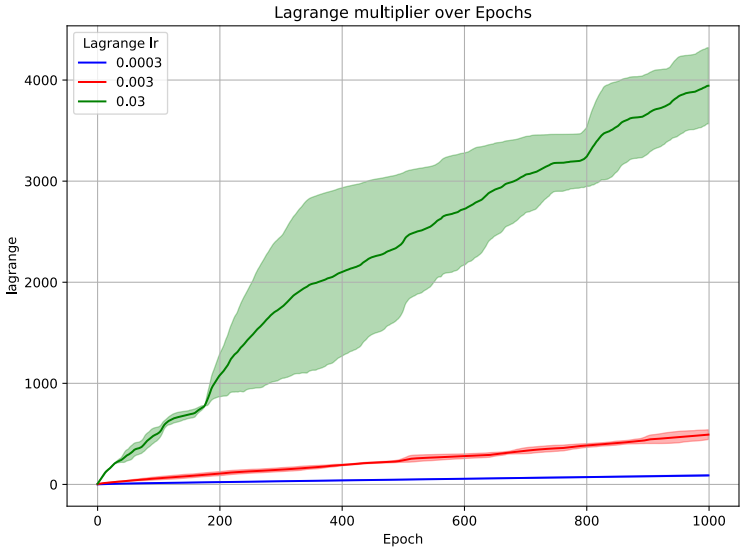


Figure 4.10: Training curves of Lagrange multiplier over epochs for different learning rate initializations in the Lagrange multiplier network

Table 4.2: Hyperparameters used in Experiments 4.3 and 4.4. In each experiment, either the Lagrange multiplier initialization bias or the learning rate was varied, as shown in the corresponding plots. The other hyperparameter was fixed to the value listed in this table.

Hyperparameter	**Value**
Environment	Point Goal
Number of epochs	1000
Number of steps per epoch	30000
Hidden layer size (MLP)	64
gamma (discount factor)	0.99
lambda (GAE)	0.97
Learning rate (policy)	3e-4
Learning rate (value function)	1e-3
Learning rate (Lagrange multiplier)	3e-4
Lagrange multiplier init (bias)	0

4.5 Evaluation Results

In this section, we present the a comparative evaluation of PPO Lagrangian, Feasible Actor-Critic, and our proposed method across two tasks: Car Goal, and Car Button.

4.5.1 Car Goal

In the Car Goal task, we compare the performance of Feasible Actor-Critic (FAC), PPO Lagrangian, and our proposed PPO Lagrangian Network (PPO-Lagnet). As shown in Fig. 4.11, all three methods converge to similar return values, indicating comparable task performance in terms of reward. In terms of cost return (Fig. 4.12), both PPO-based methods (PPO Lagrangian and PPO-Lagnet) show stable and consistent reductions in cost over time. In contrast, the FAC method produces highly

fluctuating cost values throughout training. In addition, as shown in Fig. 4.11, the FAC method exhibits higher variance in return compared to the PPO-based methods. This instability may be caused by a conflict between SAC's entropy regularization and the penalty term introduced by the Lagrange multiplier. While entropy regularization encourages exploration, the penalty term enforces constraint satisfaction, which can lead to conflicting optimization objectives. These factors make the FAC method more sensitive to hyperparameter choices and appear to result in lower stability compared to the on-policy PPO-based approaches. Finally, as shown in Fig. 4.13, we can observe that the Lagrange multiplier network in the FAC method was initialized with zero bias, as large initial values were found to cause unstable learning dynamics in preliminary experiments. Additionally, we note that this experimental result differs from the one reported in the FAC paper. As mentioned in Section 4.1, we use a dense cost formulation, where the actual cost values vary depending on the state. In contrast, the FAC implementation uses a sparse cost formulation, where the agent receives a cost of 1 if a constraint is violated and 0 otherwise. This leads to differences in the learned Q-function and cost Q-function values, potentially resulting in different optimization process and constraint enforcement behavior.

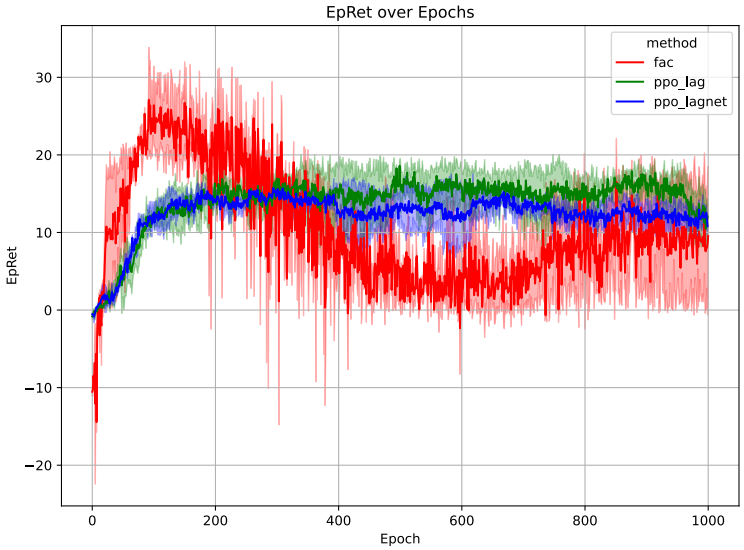


Figure 4.11: Training curves of return over epochs for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network on the Car Goal task

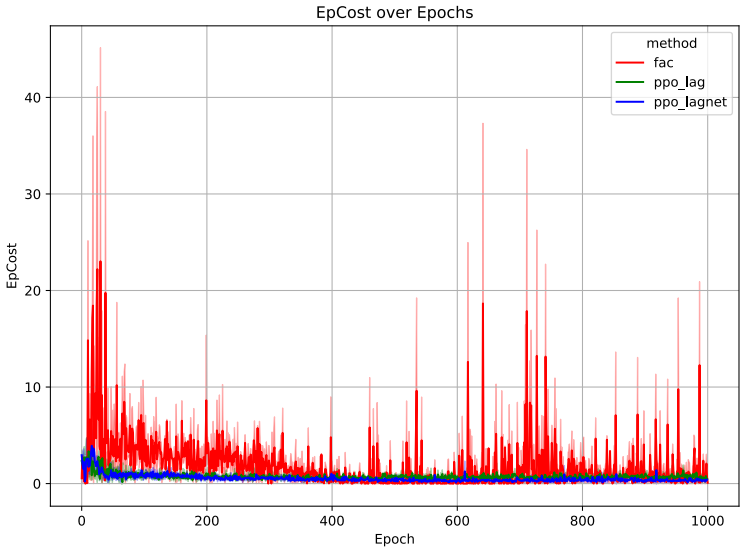


Figure 4.12: Training curves of cost return over epochs for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network on the Car Goal task

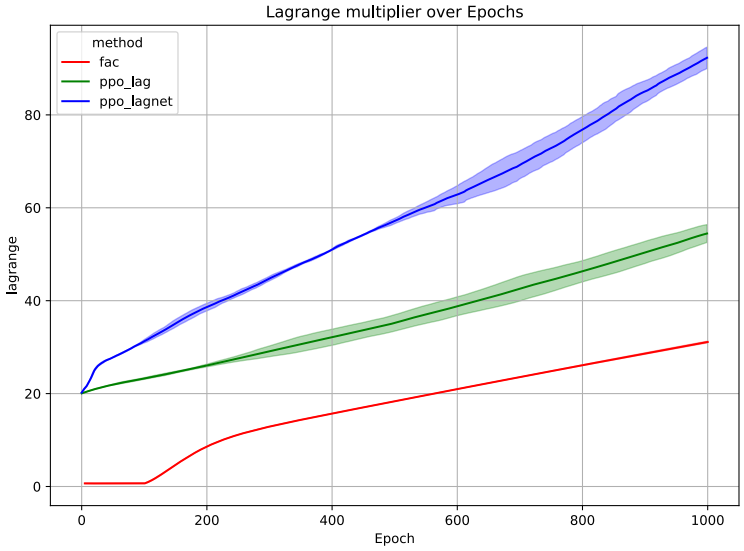


Figure 4.13: Training curves of Lagrange multiplier over epochs for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network on the Car Goal task

4.5.2 Car Button

Unlike the Car Goal task discussed in Section 4.5.1, this task contains more obstacles, which makes policy learning difficult when the cost limit is set to zero (i.e., when any constraint violation is strictly prohibited). Therefore, in this task, we set the state-wise allowable cost to 0.3, which corresponds to allowing a maximum cumulative cost of 300 per episode (given the episode length is fixed at 1000 steps). Additionally, since setting the Lagrange multiplier network's bias to 20 resulted in unstable learning in this task, we initialized it with zero bias in this experiment. As shown in Fig. 4.14, our proposed method demonstrates stable convergence in return. In contrast, the PPO Lagrangian method (green) shows unstable policy behavior between epochs 300 and 600, followed by performance recovery in the later stages. For the FAC method, the policy does not improve until around epoch 800, and only starts to receive meaningful

rewards after epoch 900. Looking at Fig. 4.15, we observe that the cost values of FAC begin to increase around the same time its return starts to improve. These results indicate that FAC needs more training iterations in this task to effectively balance reward acquisition and constraint satisfaction. Both PPO-based methods (PPO Lagrangian and our proposed method) appear to satisfy the constraints effectively, and in particular, our proposed method maintains stable constraint satisfaction starting from around epoch 100.

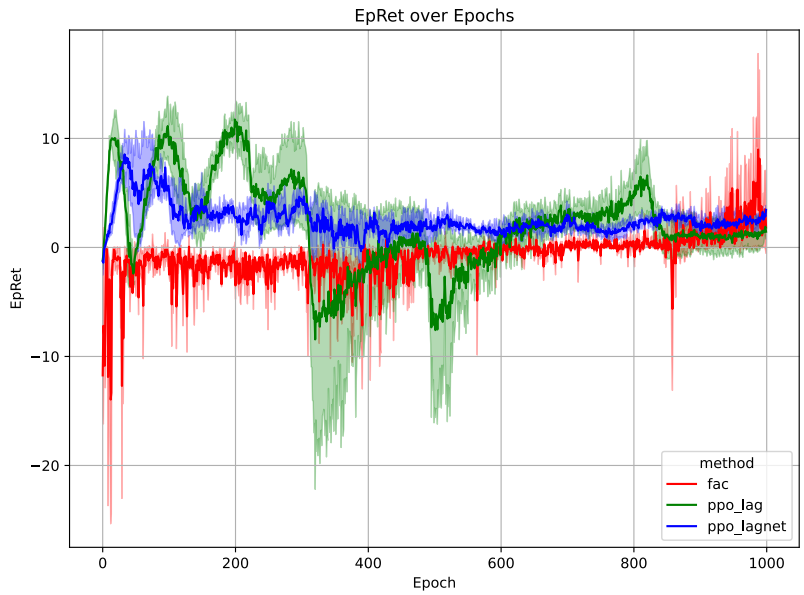


Figure 4.14: Training curves of return for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network in Car Button task

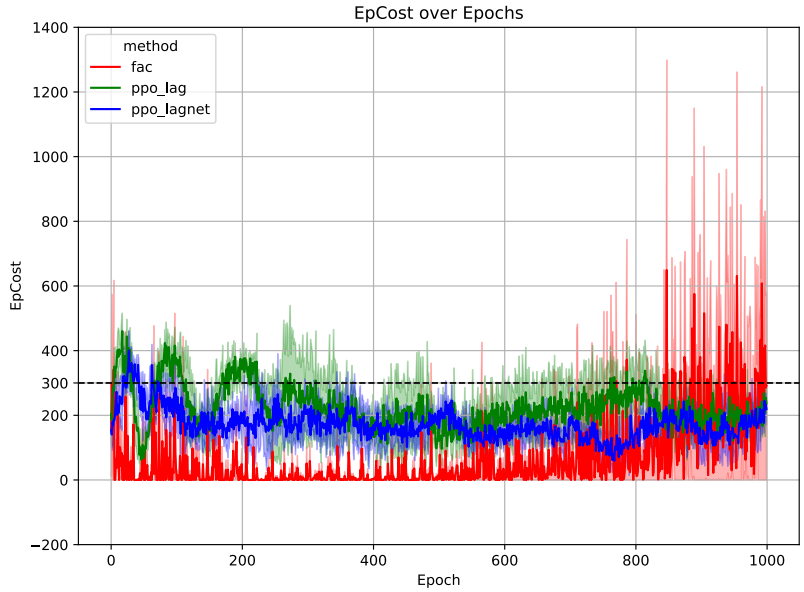


Figure 4.15: Training curves of cost return for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network in Car Button task

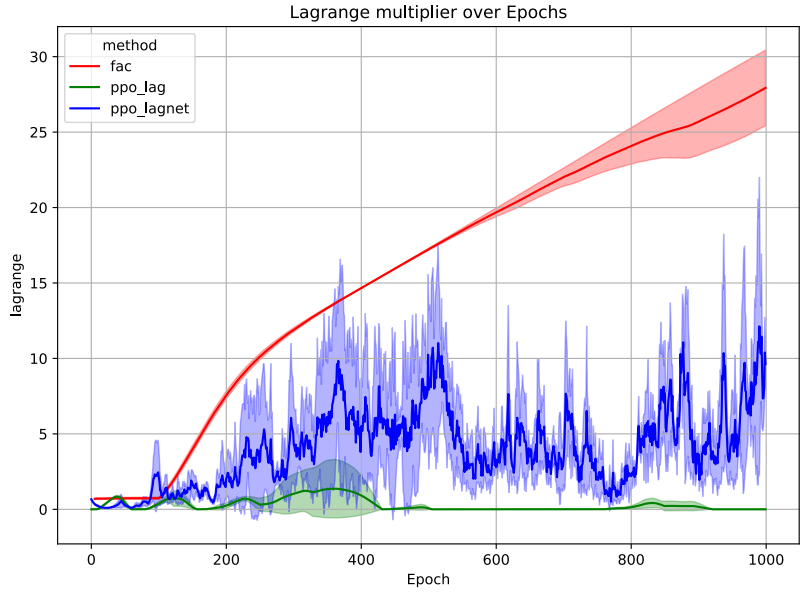
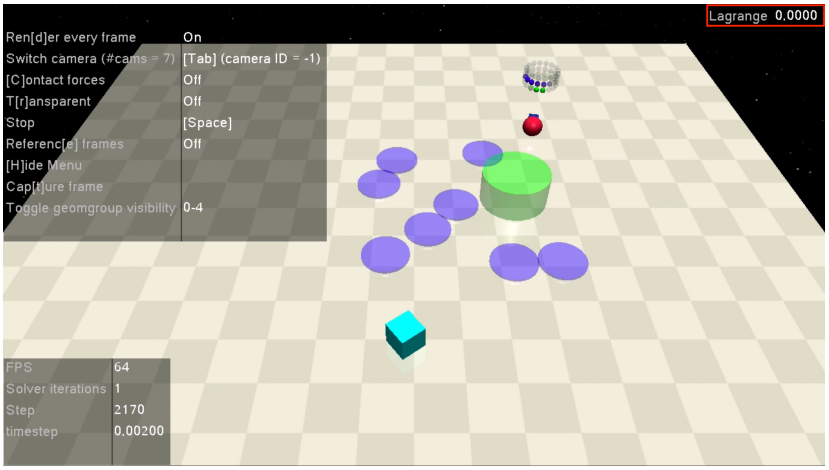


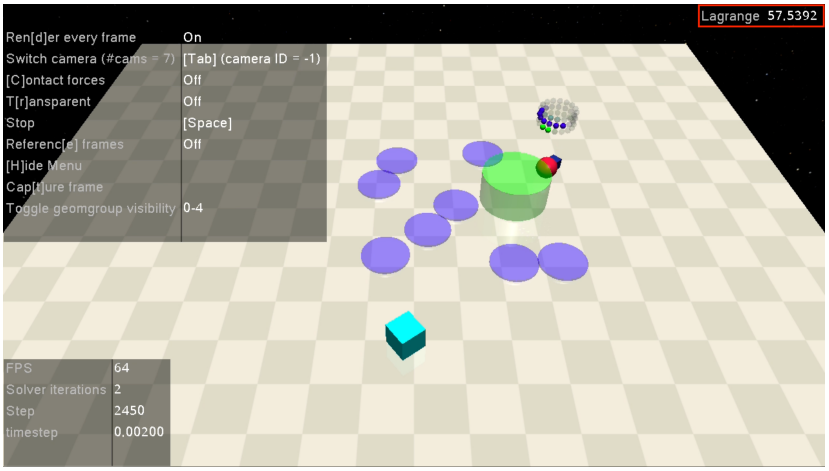
Figure 4.16: Training curves of Lagrange multiplier for PPO Lagrangian, Feasible Actor-Critic, and PPO Lagrangian Network in Car Button task

4.6 Evaluation Lagrange Multiplier Network

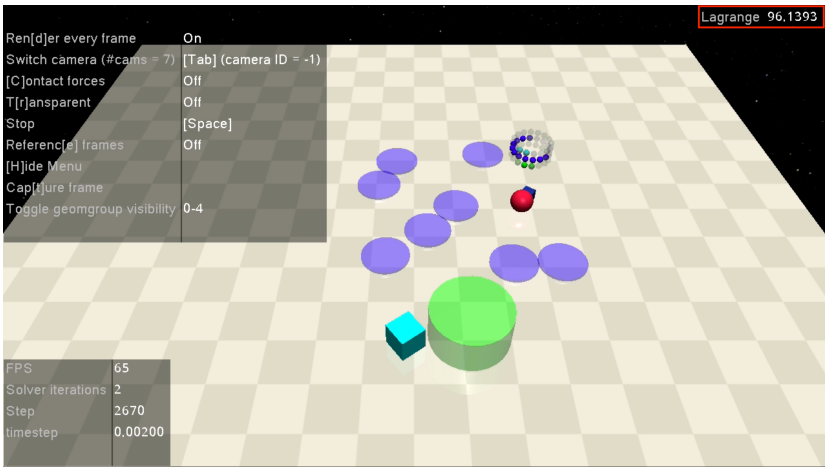
In this section, we evaluate the performance of the Lagrange multiplier network trained in the Point Goal task. We utilize the trained Lagrange multiplier network at test time, thereby enabling the assessment of whether the current state is potentially unsafe. As illustrated in Fig. 4.17 and Fig. 4.18, the trained Lagrange multiplier network outputs (displayed in the top-right corner of each figure) near-zero or low values when the agent is in a safe state, meaning there are no nearby obstacles or the agent is not moving toward hazardous regions. As the agent approaches obstacles, the output of the network gradually increases, indicating a higher level of potential risk. These results demonstrate that the Lagrange multiplier network has successfully learned to assign appropriate penalties based on the agent's proximity to unsafe areas. Therefore, it can be effectively used at test time to assess the safety of a given state.



(a) Step 2170

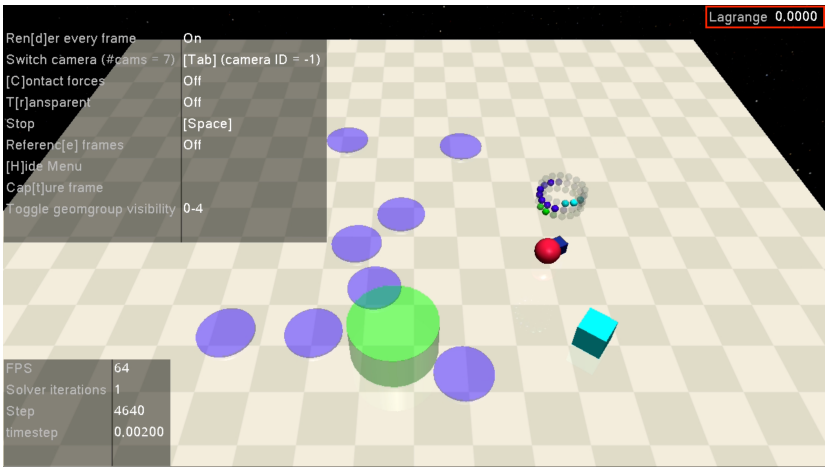


(b) Step 2450

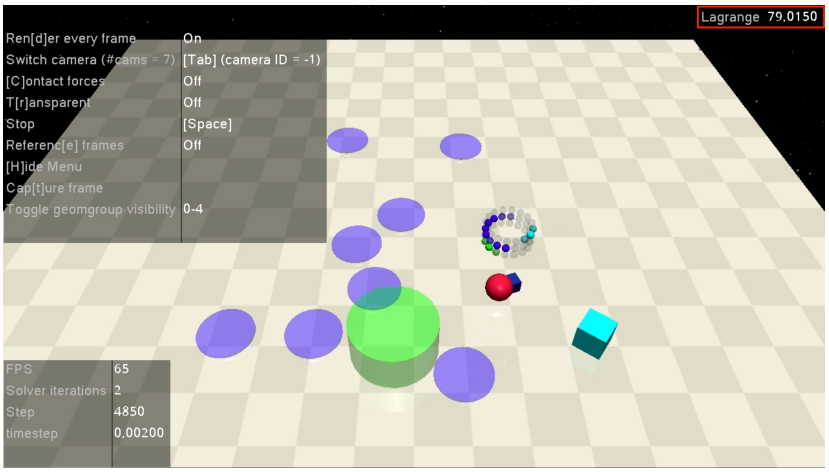


(c) Step 2670

Figure 4.17: Visualization of the Lagrange multiplier network output at different steps in the test scenario 1



(a) Step 4640



(b) Step 4850

Figure 4.18: Visualization of the Lagrange multiplier network output at different steps in the test scenario 2

Chapter 5

Conclusion

5.1 Conclusion

In this thesis, we proposed an extension to Proximal Policy Optimization (PPO) by incorporating a Lagrange multiplier network, enabling the policy to account for state-wise constraints during training. This method incorporates a network that estimates state-wise Lagrange multipliers to enforce safety constraints at the state level, allowing for more flexible and precise control over the policy's behavior in constrained environments. We conducted an empirical analysis of the impact of the bias initialization and learning rate on the performance of the Lagrange multiplier network. Our experiments show that setting the bias in the range of 0 to 20 and using a learning rate similar to that of the policy network yields the most stable and effective learning outcomes. Furthermore, when compared to existing methods such as PPO Lagrangian and Feasible Actor-Critic, our proposed approach exhibited more consistent and reliable constraint satisfaction across different tasks. These results suggest that our method provides a robust framework for safe reinforcement learning under state-wise constraints. We also found that the trained Lagrange multiplier network can be deployed at test time to assess whether the current state is potentially unsafe. If the network outputs a high value for a given state, it typically indicates that the agent is near a constraint-violating

region (e.g., close to an obstacle). In such cases, we observed that the agent adjusts its behavior to avoid hazards and navigates more safely.

5.2 Limitations and Future Work

While our proposed method demonstrates promising results in enforcing state-wise constraints, several limitations remain. First, since the policy is trained to satisfy constraints by incorporating penalty terms, it does not offer deterministic safety guarantees. Even after training, constraint violations may still occur. To overcome this issue, the trained Lagrange multiplier network can be leveraged at test time to identify high-risk situations. If an action is considered unsafe, a safety filter may be used to override it, thereby improving safety at deployment. Moreover, while our method does not guarantee stability during training, this limitation may be addressed by integrating techniques that ensure safe exploration. For example, projection-based approaches [20, 21], which optimize the policy while keeping it within the feasible region, could be considered. Second, our current evaluation is restricted to the relatively simple Safety Gym environments implemented using the MuJoCo engine [22]. To further validate the practicality and generalizability of our approach, we plan to extend our experiments to more realistic and safety-critical domains, such as CARLA, a high-fidelity autonomous driving simulator. This transition would enable us to evaluate the robustness, scalability, and real-world applicability of the proposed method in more complex and dynamic scenarios.

Summary

State-wise Safety in Autonomous Driving

via Lagrangian-based

Constrained Reinforcement Learning

This thesis studies how to train policies to satisfy state-wise safety using state-wise constrained reinforcement learning. We focus on Lagrangian-based approaches and identify their key limitations, particularly with respect to stability and convergence. To address these challenges, we propose PPO Lagrangian Network, an extension of Proximal Policy Optimization that incorporates a Lagrange multiplier network to dynamically enforce safety constraints at the state level. Extensive experiments on the OpenAI Safety Gym benchmark show that proposed method achieves more stable and reliable constraint satisfaction than existing approaches such as PPO Lagrangian and Feasible Actor-Critic (FAC). We further investigate how hyperparameter choices, such as the initialization bias and learning rate of the Lagrange multiplier network, affect performance. Our findings indicate that careful tuning of these parameters is crucial for maintaining stability under hard constraint settings. Additionally, we demonstrate that the trained Lagrange multiplier network can be reused at test time as a risk estimator, allowing the agent to assess the safety of a given state and adjust its behavior accordingly.

References

1. R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

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

3. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

4. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

5. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

6. C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

7. O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

8. W. Zhao, T. He, R. Chen, T. Wei, and C. Liu, "State-wise safe reinforcement learning: A survey," *arXiv preprint arXiv:2302.03122*, 2023.

9. M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

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

11. J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

12. T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

13. T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

14. P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

15. E. Altman, *Constrained Markov decision processes*. Routledge, 2021.

16. A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," *arXiv preprint arXiv:1910.01708*, vol. 7, no. 1, p. 2, 2019.

17. H. Ma, Y. Guan, S. E. Li, X. Zhang, S. Zheng, and J. Chen, "Feasible actor-critic: Constrained reinforcement learning for ensuring statewise safety," *arXiv preprint arXiv:2105.10682*, 2021.

18. J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang, "Safety gymnasium: A unified safe reinforcement learning benchmark," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

19. Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *The 30th international joint conference on artificial intelligence (ijcai)*, 2021.

20. J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *International conference on machine learning*, pp. 22–31, PMLR, 2017.

21. T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge, "Projection-based constrained policy optimization," *arXiv preprint arXiv:2010.03152*, 2020.

22. E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012.

Acknowledgements

special thanks to...

모두들 감사합니다.