DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master Thesis

Safe Multi-Agent Reinforcement Learning with Bi-level Optimization in Autonomous Driving

Author:	Zhi Zheng
Supervisor:	Prof. Dr. -Ing. Alois Knoll
Advisor:	Shangding Gu
Submission Date:	30.06.2023

I confirm that this master thesis is my own work and I have documented all sources and material used.

Munich, 30.06.2023 Zhi Zheng

Zhi Zheng

Acknowledgments

I want to convey my sincere thanks to my supervisor Prof. Dr. -Ing. Alois Knoll and advisor M.Sc. Shangding Gu for giving me the opportunity to write this thesis and for providing detailed guidance in both my research and personal life.

I would also like to express my gratitude to my parents and friends, who have always been willing to accompany and support me in pursuing my Master's degree.

Abstract

This thesis explores integrating safe reinforcement learning techniques into autonomous driving scenarios. The primary objective is to incorporate the Bi-level problem(Stackelberg model) with Q-learning algorithms and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithms, leveraging the Lagrangian method to enforce safety constraints. The focus is developing an approach that ensures safe interactions between multiple autonomous vehicles while optimizing their objectives.

This task focuses mainly on scenarios involving only two vehicles, where one car acts as the leader and the other acts as the follower. The leader is the agent that has access to the follower agent's policy and the ability to commit to action before the follower. The follower can observe actions taken by the leader and respond to maximize his payoff. We propose two new safe reinforcement learning algorithms to address this problem: Constrained Stackelberg Q-learning for discrete action space and Constrained Stackelberg MADDPG for continuous action space.

We evaluate our algorithms on multiple mixed cooperative and competitive multi-agent environments customized through the existing highway environments. The results demonstrate the efficacy of the proposed approaches in enhancing safety while achieving satisfactory performance. Comparative analysis against existing methods and frameworks in the field further highlights the advantages and contributions of the proposed algorithms.

Keywords: Safe reinforcement learning, Stackelberg model, Bi-level, Multi-Agent Deep Deterministic Policy Gradient (MADDPG), Lagrangian method, Autonomous driving.

Contents

1 Introduction

1.1 Motivation

In the field of multi-agent reinforcement learning (MARL), the Stackelberg model has gained significant attention and has been widely studied. Many research papers have shown that incorporating the Stackelberg model can substantially improve algorithm performance, especially in complex decision-making scenarios. However, while the focus has primarily been on enhancing performance, the issue of safety has often been overlooked. This is where safe reinforcement learning techniques play a crucial role.

In traditional MARL algorithms, there is a lack of communication and cooperation strategies among agents. Each agent's goal is to maximize its utility, which may lead to achieving Nash equilibrium [1]. However, in dynamic or complex environments, the actions and policies of other agents may change over time, leading to non-stationarity. This makes it challenging to learn optimal policies and coordinate actions effectively. The Stackelberg model provides a framework for hierarchical decision-making, allowing agents to assume leader and follower roles. The leader agent, known as the Stackelberg leader, can anticipate and influence the actions of the follower agents, known as Stackelberg followers. This hierarchical structure enables the leader to optimize his objectives while accounting for the followers' responses.

The application of the Stackelberg model in MARL has demonstrated remarkable improvements in various domains, such as autonomous driving [2] [3], robotics, and multi-player games [4] [5]. By leveraging the Stackelberg model, agents can exhibit more sophisticated strategies, better coordination, and increased overall performance. These advancements have opened up new avenues for research and applications in complex multi-agent scenarios.

However, amidst the excitement surrounding the performance improvements achieved with the Stackelberg model, the crucial aspect of safety has often been overlooked. In many real-world applications, ensuring safety is of paramount importance [6] [7]. Autonomous driving, for example, involves multiple vehicles interacting within a shared environment, where safety is a critical concern. Neglecting safety considerations can lead to hazardous situations and potentially catastrophic consequences.

To address this gap, the field of safe reinforcement learning has emerged, focusing on the integration of safety considerations into reinforcement learning algorithms. By incorporating safety as an explicit objective, these techniques aim to balance optimizing performance and ensuring safe actions within multi-agent systems. Given that maximizing the reward alone may not effectively address rare but significant negative outcomes, alternative criteria are required to evaluate safety and risk in reinforcement learning settings. To address this challenge, the objective of reward maximization can be modified to incorporate risk considerations using the Risk-Sensitive criterion [8] or its constrained criterion [9].

This thesis aims to bridge the gap between the Stackelberg model and safety in the context of MARL, explicitly focusing on autonomous driving scenarios. By incorporating safe reinforcement learning techniques within the Stackelberg framework, we seek to enhance the agents' performance while guaranteeing safety in complex and dynamic environments. Through the utilization of safety-aware algorithms, such as the Lagrangian method, we can enforce safety constraints and promote cautious decision-making among autonomous agents. By integrating the strengths of the Stackelberg model and safe reinforcement learning, we try to discover the potential for advanced and safe coordination strategies in multi-agent systems.

1.2 Summary

In the following chapters, this thesis explores the integration of safe reinforcement learning techniques into autonomous driving scenarios, with a focus on incorporating a Stackelberg model with Q-learning and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithms, using the Lagrangian method to enforce safety constraints.

Chapter 2 provides in-depth background on reinforcement learning (RL), introducing key concepts such as Markov Decision Processes (MDPs), Q-learning, and policy gradients. It establishes the theoretical foundation necessary to understand the algorithms and methodologies used in this thesis.

Chapter 3 delves into the related work on safe reinforcement learning, highlighting existing approaches, frameworks, and techniques to address safety concerns in autonomous systems. It also explores the Stackelberg game theory and its connection to RL, setting the stage for the proposed integration of the Stackelberg model with reinforcement learning algorithms.

Chapter 4 presents the novel algorithms developed in this thesis. It details the implementation of the Stackelberg model in the Q-learning and MADDPG framework and the application of the Lagrangian method to ensure safe interactions between autonomous vehicles. The algorithms' key components, optimization strategies, and safety constraints are thoroughly explained.

In Chapter 5, we conduct a comprehensive convergence analysis of our proposed algorithms. We try to prove the convergence properties of the algorithms using fixed-point theory and other mathematical techniques. This analysis provides insights into the stability and convergence of our algorithms.

Chapter 6 focuses on the experiments conducted to evaluate the proposed algorithms' performance and presents the experiments' results. It describes the simulation environment specifically designed to simulate various autonomous driving scenarios. The evaluation metrics are discussed in detail, including success rates, accumulated rewards, convergence analysis, and prioritization of vehicles reaching their destinations. It also compares the performance of the proposed algorithms against existing methods and frameworks in safe reinforcement learning. The insights gained from the experiments are discussed, highlighting the advantages and novel contributions of the proposed approach.

Chapter 7 concludes the thesis by summarizing the research's key findings, implications, and contributions. It reflects on the effectiveness of the proposed algorithms in improving safety while maintaining satisfactory performance in autonomous driving scenarios. Additionally, future research directions and potential areas of improvement are identified.

2 Background

2.1 MDP and constrained MDP

2.1.1 Markov decision process

The Markov Decision Process (MDP) is a mathematical framework utilized for modeling decision-making in dynamic and uncertain environments [10]. It involves a sequential interaction between an agent and its environment, where the agent receives feedback in the form of rewards or penalties based on the state transitions and actions taken. The agent's goal is to learn an optimal policy, a collection of strategies or rules, that maximizes the expected total reward over time. By employing the MDP framework, decision-making processes can be effectively modeled and analyzed, enabling the development of intelligent algorithms for a wide range of applications.

The MPD is usually represented by a tuple (S, A, R, P, γ), where:

- S represents the state space,
- A represents the action space,
- $R = S * A * S \to \mathbb{R}$ represents the reward function,
- $P = S * A * S \to [0,1]$ represents the transition probability function, $P(s'|s,a)$ gives the transition probability to state s' under the previous state s and the action a,
- $\gamma \in [0,1]$ is the discount factor, which influences the relative significance of future rewards in comparison to immediate rewards.

A policy π is a map from state s to probability distributions over actions a. The aim of reinforcement learning is to find a policy that can maximize the expected accumulated discounted reward $J(\pi) = \mathbb{E}_{\tau\sim\pi} \sum_{t=1}^{\infty} \gamma^t R(s^t, a^t, s^{t+1})$, where $\tau \sim \pi$ denotes the trajectory produced by the agent by following policy π. Therefore, the learning problem of MDP can be formulated as an optimization problem:

$$\max_{\pi} \quad \mathbb{E}_{\tau\sim\pi} \sum_{t=1}^{\infty} \gamma^t R(s^t, a^t, s^{t+1}) \tag{2.1}$$

2.1.2 Constrained Markov decision process

Constrained Markov Decision Process (CMDP) [11] entends MDP with constraints on acceptable policies. CMDP augment MDP with auxiliary cost functions $C_1, C_2 ... C_n$ and limits $d_1, d_2 ... d_n$, where $C_i = S * A * S \rightarrow \mathbb{R}$. Similar to reward, $J_{C_i}(\pi) = \mathbb{E}_{\tau \sim \pi} \sum_{t=1}^{\infty} \gamma^t C_i(s^t, a^t, s^{t+1})$ denotes the expected accumulated discounted cost when an agent acts following policy π. Therefore, the learning problem of CMDP can be formulated as follow:

$$\begin{aligned} \max_{\pi} \quad & \mathbb{E}_{\tau \sim \pi} \sum_{t=1}^{\infty} \gamma^t R(s^t, a^t, s^{t+1}) \\ \text{s.t.} \quad & \mathbb{E}_{\tau \sim \pi} \sum_{t=1}^{\infty} \gamma^t C_i(s^t, a^t, s^{t+1}) \leq d_i \end{aligned} \tag{2.2}$$

2.2 Reinforcement learning

Reinforcement learning focuses on developing algorithms capable of learning optimal decision-making strategies by interacting with an environment. RL draws inspiration from the way humans learn through trial and error, gradually refining their actions to maximize cumulative rewards. The reinforcement learning process is usually modeled as a standard Markov Decision Process (MDP) or a Partially Observable Markov Decision Process (POMDP) [12], depending on whether the agent has complete or partial knowledge of the environment, like a single agent or multi-agent scenario. RL algorithms encompass various approaches, including value-based methods such as Q-learning and SARSA, policy-based methods like PPO (Proximal Policy Optimization) and TRPO (Trust Region Policy Optimization), and model-based methods such as Monte Carlo tree search.

2.2.1 Value functions and Bellman equations

In RL, the ultimate objective is to learn an optimal policy that maximizes the expected return $J(\pi)$, which can be measured by either infinite-horizon discounted or finite-horizon undiscounted rewards. To assess and evaluate the performance of a given policy, certain criteria, known as value functions, need to be defined. Two commonly used definitions of value functions are as follows:

- State-Value function $V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi} \sum \gamma^t R(\tau | s_0 = s)$, which refers to the anticipated cumulative rewards obtained when starting from a particular state, s, and consistently following a given policy, π.

- Action-Value function $Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi} \sum \gamma^t R(\tau|s_0 = s, a_0 = a)$, which gives the expected return if you start from state s and take action a (can be chosen freely inside the action space), and then consistently act according to policy π. If the first action were to be chosen according to policy π, these two value functions can be written inequality: $V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi} Q^{\pi}(s,a)$

These two value functions can be rewritten into recursive forms according to the Bellman equation [13]:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi, s'\sim P}(R(s,a) + \gamma V^{\pi}(s')) \tag{2.3}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s'\sim P}(R(s,a) + \gamma \mathbb{E}_{a'\sim\pi} Q^{\pi}(s',a')) \tag{2.4}$$

The next state s' follows the transition probability distribution $P(s)$ and the next action a' follows the policy $\pi(s')$. The fundamental concept underlying the Bellman equation is that the value of a particular state is determined by the sum of the immediate reward obtained at that state and the value of the subsequent state it transitions to.

The optimization problem is defined as follows corresponding to the Bellman optimality equations to achieve the best performance and find the optimal policy.

$$V^{*}(s) = \max_a \mathbb{E}_{s'\sim P}(R(s,a) + \gamma V^{*}(s')) \tag{2.5}$$

$$Q^{*}(s,a) = \mathbb{E}_{s'\sim P}(R(s,a) + \gamma \max_{a'} Q^{*}(s',a')) \tag{2.6}$$

By following the Bellman optimality equations, the algorithm is able to converge and yield optimality.

2.2.2 Q-learning and Deep-Q-learning

Q learning [14] reinforcement learning algorithm whose purpose is to learn the optimal action-value function Q^* through training and the above-mentioned Bellman optimality equation. The action selection of Q-learning is not executed through a parameterized policy function but from solving an optimization problem for a given state s:

$$a = \max_a Q^{*}(s,a) \tag{2.7}$$

As for a simple environment with discrete state and action spaces, Q-learning is usually implemented as tabular Q-learning, where the Q-values are stored in a matrix-like table. Each row represents a state, and each column represents an action. However, this approach may become impractical or infeasible when dealing with problems that have large state and action spaces.

To overcome the difficulty of a complex environment using large continuous state and action spaces, the value function is usually approximated using neural networks(known as Deep Q Network) [15]. The algorithm learns an approximation of the optimal function, denoted as Q^*, corresponding to the optimal policy by minimizing the TD error using gradient descent techniques:

$$L(\phi) = \mathbb{E}_{(s,a,r,d,s')\sim D}[(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_{\phi targ}(s',a')))^2] \tag{2.8}$$

The function Q_ϕ and $Q_{\phi targ}$ are both Q functions and have the same structure. $Q_{\phi targ}$ is the target function used to stabilize the training. The parameter $\phi targ$ of $Q_{\phi targ}$ will be updated periodically with the latest ϕ:

$$\phi^{k+1} targ \leftarrow \phi^k \tag{2.9}$$

The trajectory (s,a,r,d,s') is sampled from the trajectory replay buffer D, which is a set of previous experiences. A large replay buffer that contains a wide range of experiences is also able to stabilize the training.

2.2.3 Policy Gradient Algorithms

Policy gradient algorithms like PPO(Proximal Policy Optimization) [16] and TRPO(Trust Region Policy Optimization) [17] are usually on-policy algorithms. In on-policy algorithms, the state value function V^π is estimated after running a whole episode of a scenario according to policy π by calculating the return $R^t = \sum_{i=t}^{T} \gamma r^i$. Similar to Deep-Q-learning, the approximation function of V is learned by minimizing the mean square error using gradient descent techniques:

$$L(\phi) = \frac{1}{|D_k|T} \sum_{\tau \in D_K} \sum_{t=0}^{T} (V_\phi(s^t) - R^t)^2 \tag{2.10}$$

where D denotes the set of trajectories sampled by following policy π. The key feature of this kind of algorithm is that they don't use old data to train the policy. The estimated value function can evaluate the performance of the policy, which increases the sample efficiency. But as a trade-off, the stability of this kind of algorithm is easily affected by the sample variance.

Denoting the policy network with parameter θ as π_θ and the expected return of the policy as $J(\pi_\theta)$, the gradient of $J(\pi_\theta)$ can be expressed as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta log \pi_\theta(a^t|s^t) A^{\pi_\theta}(s^t,a^t)] \tag{2.11}$$

where τ is the trajectory samples and A^{π_θ} represents the advantage function. The advantage function quantifies the relative advantage of taking a specific action in a state s, compared to randomly selecting an action based on the policy $\pi_\theta(\cdot|s)$ and consistently following it thereafter. The advantage function can be defined as follow:

$$A^{\pi_\theta} = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{2.12}$$

2.2.4 Actor-Critic Method

Actor-Critic methods, such as DDPG (Deep Deterministic Policy Gradient) [18], are algorithms widely used in reinforcement learning. The approach can be characterized as a fusion of Q-learning and Policy Gradient algorithms. In this approach, the agent learns a value function(Critic) and a policy(Actor) simultaneously, using the value function to guide policy updates. By this means, Q-learning algorithms are able to be implemented in the continuous action space.

Critic

The so-called Critic refers to the commonly used Q function in Q-learning, which is approximated by a neural network Q_ϕ. Known from the Bellman optimality equation and Deep-Q-learning algorithm, The Q function Q_ϕ is learned through minimizing the TD-error defined in 2.8 with the help of $Q_{\phi targ}$. However, unlike discrete action space, the max operation in continuous action space is quite challenging. To address this problem, DDPG uses a target policy network $\pi_{\theta targ}$ with parameter $\theta targ$ to approximately compute the best action $a = \pi_{\theta targ}(s)$ to maximize $Q_{\phi targ}$ for state s. Therefore, the Q-learning is performed by minimizing the following TD error using the gradient descent technique:

$$L(\phi) = \mathbb{E}_{(s,a,r,d,s')\sim D}[(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_{\phi targ}(s', \pi_{\theta targ}(s'))))^2] \tag{2.13}$$

Actor

The Actor is a policy function that can directly output an action a for state s. To stabilize the learning process, both the policy function and the exploration policy network are approximated using neural networks. The policy function is denoted as π_θ, with parameter θ, while a target policy network, which shares the same structure as the policy network, is employed during training. This approach helps maintain stability and facilitates effective learning. The goal of the training is to learn a policy that computes an action that can maximize the value function $Q_\phi(s,a)$. It is quite obvious from the forward computation graph $Q_\phi(s, \pi_\theta(s))$, the gradient of Q_ϕ with respect to θ

can be directly computed under the assumption that the value function is differentiable with respect to action a. Therefore, gradient ascent can be applied to solve the following problem:

$$\theta = \arg\max_{\theta} \mathbb{E}_{s \sim D}(Q_\phi(s, \pi_\theta(s))) \tag{2.14}$$

target network

As mentioned above, there are two target networks in Actor-Critic algorithms, which are target policy network $\pi_{\theta targ}$ and target Q-function network $Q_{\phi targ}$. The target networks are only used in formula 2.13 when we minimize the TD error. Assuming that without the help of target networks, the two parts in the mean square error function actually have the same parameters. Both parts are changing simultaneously when performing minimization, like someone is chasing himself, which is hard to achieve convergence. To address this challenge, a solution involves utilizing a second network known as the target network, which introduces a time delay and approximates a set of parameters close to ϕ.

In contrast to the conventional Q-learning algorithm, which periodically copies the main network to update the target network, the update rule for target networks in this approach is different. In Actor-Critic algorithms, the target network is updated at each training step using a technique called Polyak averaging:

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1 - \rho)\phi \tag{2.15}$$

$$\theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta \tag{2.16}$$

where ρ is the update hyperparameter.

2.3 Multi-agent reinforcement learning

Although many excellent reinforcement learning algorithms have emerged recently, they are mostly limited to single-agent scenarios. As the complexity of problems increases, the algorithms have to manage the interaction between multiple agents. For example, multiple vehicles autonomous driving, multiple robot control, and some multiplayer games. Unfortunately, the performance of traditional reinforcement learning algorithms is not ideal when transplanted to multi-agent settings. The main reason is the introduction of multiple agents, where each agent not only has to react to different environments but also to the different actions of other agents. This makes it difficult for the entire environment to reach a static equilibrium.

2.3.1 Game theory

Game theory is introduced to reinforcement learning to address the issue of reaching static equilibrium. Game theory is a natural and useful framework for modeling the interaction and decision-making among multiple agents in a reinforcement learning setting [19]. Each agent is assumed to be rational and strategic so that they all adhere to a pre-designed strategy. This can optimize their expected rewards and push all the agents to an equilibrium.

Besides, the relationship between agents can also be different depending on different scenarios, for example, cooperative, competitive, and mixed cooperative-competitive. Game theory can provide a set of powerful tools and concepts to study and analyze these scenarios. In other words, by studying how humans react in these scenarios, we can enable machines to learn how to behave similarly.

2.3.2 Nash Q-learning

Nash Q-learning [1] is an extension of the Q-learning algorithm for multi-agent scenarios. It seeks to find a Nash equilibrium in a non-cooperative and general-sum game under the guidance of game theory.

Adapting Q-learning to multi-agent scenarios requires agents to go beyond perceiving only their own actions and the environment. They need to possess a comprehensive understanding of the actions taken by other agents in order to effectively respond to their behaviors. For this reason, the Q function of any individual agent $i = 1, 2...n$ should take the joint action $a = (a_1, a_2...a_n)$ into account. The new Q function $Q_i(s, a_1, a_2...a_n)$ for agent i is the joint state action-value function, which evaluates the expected long-term reward for each combination of actions all agents take. Similarly, the value function can also be constructed in this way $V_i^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}(\sum_{t=0}^{T} \gamma R_i^t(s))$, where $\pi = (\pi_1, \pi_2...\pi_n)$ denotes the joint strategy. A Nash equilibrium point is defined as a joint strategy tuple $\pi^* = (\pi_1' \pi_2' \ldots, \pi_n^*)$ that satisfies the following condition for all agents $i = 1, 2, \ldots, n$ and all states $s \in S$:

$$V_i(s, \pi_1^*, \pi_2^*...\pi_n^*) \geq V_i(s, \pi_1^*...\pi_{i-1}^*, \pi_i, \pi_{i+1}^*...\pi_n^*) \quad \pi^i \in \Pi^i \tag{2.17}$$

It means that each agent's policy in Nash equilibrium is the best response to the other agents' optimal policies. No agent is incentivized to change their policy because any tweak gives less reward. The optimal Q function under Nash equilibrium can be therefore defined as follow as Nash Q:

$$Q_i^*(s, a_1, a_2...a_n) = r_i(s, a_1, a_2...a_n) + \gamma \sum p(s'|s, a_1, a_2...a_n) V_i(s', \pi_1^*, \pi_2^*...\pi_n^*) \tag{2.18}$$

where $p(s'|s, a_1, a_2...a_n)$ is the transition probability. The Nash Q-Value signifies the anticipated long-term reward of an agent, assuming all agents adhere to a joint Nash equilibrium strategy following a joint action.

Under these definitions, the iterative update rule for Nash Q Learning is defined as follows:

$$Q_i^{t+1}(s, a_1, a_2...a_n) = (1-\alpha)Q_i^t(s, a_1, a_2...a_n) + \alpha(r_i^t + \gamma NashQ_i^t(s')) \tag{2.19}$$

$$NashQ_i^t(s') = \pi_1(s')\pi_2(s')...\pi_n(s')Q_i^t(s') \tag{2.20}$$

where $\alpha \sim [0,1)$ is the learning rate sequence. $NashQ_i^t(s)$ stands for the agent i's payoff under state s and the found Nash equilibrium point. In complex situations, there may be multiple Nash equilibrium points, which can cause the specific update process to differ depending on the Nash equilibrium point found.

2.3.3 Multi-Agent Deep Deterministic Policy Gradient

As mentioned, traditional single-agent algorithms perform poorly when extended to multi-agent scenarios. The environment cannot achieve a stationary point from the perspective of an individual agent while other agents' policies are constantly changing. The MADDPG algorithm [20] addressed this problem by adopting the framework called Centralized-training-Decentralized-execution. The following two sections will describe this framework with the MADDPG algorithm.

Centralized-training

Similar to Nash Q-learning, the Q function of each agent in MADDPG is extended to a centralized Q function that considers all agents' actions and observations. $Q_i^\pi(x, a_1, a_2...a_n)$ denotes the centralized action-value function, where $\pi = (\pi_1, \pi_2...\pi_n)$ is the set of all agents' policies parameterized by $\theta = (\theta_1, \theta_2...\theta_n)$ and $x = (o_1, o_2...o_n...)$ consists of the observation from all agents and some additional information if available. The expression for the gradient of the expected return for agent i can be formulated as follows:

$$\bigtriangledown_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p, a_i\sim\pi_i}[\bigtriangledown_{\theta_i} log\pi_i(a_i|o_i)Q_i^\pi(x, a_1, a_2...a_n)] \tag{2.21}$$

This formula can also be extended for deterministic policies from the above stochastic policy case.

$$\bigtriangledown_{\theta_i} J(\theta_i) = \mathbb{E}_{x,a\sim D}[\bigtriangledown_{\theta_i}\pi_i(a_i|o_i) \bigtriangledown_{a_i} Q_i^\pi(x, a_1, a_2...a_n|a_i = \pi_i(o_i))] \tag{2.22}$$

where D is the replay buffer containing all agents' transition information. The update rule for MADDPG follows a similar principle to Q-learning algorithms, where it aims to minimize the TD (Temporal Difference) error.

$$
\begin{aligned}
&L(\phi_i) = \mathbb{E}_{(x,a,r,d,x')\sim D}(Q_i^{\pi} - y)^2 \\
&y = r_i + (1-d)\gamma Q_i^{\pi targ}(x', a'_1, a'_2 ... a'_n) \\
&a'_j = \pi_j^{targ}(o'_j) \quad j = 1, 2...n
\end{aligned}
\tag{2.23}
$$

where $\pi^{targ} = (\pi_1^{targ}, \pi_2^{targ} ... \pi_n^{targ})$ denotes the set of all agents' target policies with delayed parameters $\theta^{targ} = (\theta_1^{targ}, \theta_2^{targ} ... \theta_n^{targ})$. By this means, each agent contains all agents' transition information. Their reward can be structured arbitrarily to represent a cooperative, competitive, or mixed game.

Decentralized-execution

After training is completed, only the local policies are used in the execution phase. Since the local policies have been trained to take the actions of other agents into account, the agents can still achieve the global goal without explicit coordination or communication with other agents during execution. Each agent's policy π_i(short for π_{θ_i}) only takes its observation o_i as input.

$$
a_i = \pi_i(o_i) \quad i = 1, 2...n \tag{2.24}
$$

The use of the Centralize-Training-and-Decentralized-Execution framework in MADDPG has several advantages. First, it allows agents to learn how to behave in different games, like cooperative games, by considering the actions of all agents and the resulting joint effects on the environment. This can lead to better overall performance and higher rewards than policies learned independently.

Second, the centralized training process can help alleviate non-convergence in MARL settings, where each agent's policy constantly changes. MADDPG can stabilize the learning process by learning a centralized value function and prevent agents from overfitting or underfitting the environment.

Finally, the framework simplifies the learning process and reduces the computational overhead compared to other approaches, such as independent learning or communication-based methods. This is because the agents only need to train their value function and policy rather than exchanging large amounts of information with each other.

3 Related Work

3.1 Safe reinforcement learning

As reinforcement learning is increasingly applied in practice, more and more attention is being paid to its safety criterion. In some environments, the agent's safety is quite important, such as in autonomous driving and robotic platforms. The trial and error cost in these environments is extremely high. Once collisions or other events occur, they may cause high economic losses or even personal injury. Due to this reason, researchers are increasingly focusing on not only maximizing long-term rewards but also on preventing damage or harm. [6], [21].

3.1.1 Reward Constrained Policy Optimization

Reward Constrained Policy Optimization (RCPO) is a reinforcement learning framework designed to optimize policies under specified constraints with the help of the Lagrangian approach[22]. The fundamental objective of safe reinforcement learning is to address the challenges posed by constrained Markov Decision Processes (MDPs). The Lagrange relaxation technique [23] is a commonly used approach for solving problems of this type. The CMDP problem can be converted into an unconstrained problem using Lagrange relaxation. The unconstrained relaxation of the CMDP problem listed in 2.2 can be reformed as follow:

$$\max_{\theta} \min_{\lambda \geq 0} L(\theta, \lambda) = J_R(\pi_\theta) - \lambda(J_C(\pi_\theta) - d) \tag{3.1}$$

The constraint function is incorporated into the objective function by introducing a penalty term. As the value of λ increases, the solution to equation 3.1 converges to the solution of equation 2.2. This approach suggests a two-timescale process: on a faster timescale, the optimal value of θ is determined by solving equation 3.1, while on a slower timescale, λ is progressively increased until the constraint is satisfied.

Directly solving this problem is not practical because MDP problems are often enormous and complex in scale. In RCPO, it proposes an Actor-Critic algorithm to solve this problem. The actor and the Lagrangian multiplier are updated as follows:

$$\begin{aligned} \lambda_{k+1} =& \Gamma_\lambda[\lambda_k - \eta_\lambda(k) \bigtriangledown_\lambda L(\theta_k, \lambda_k)] \\ \theta_{k+1} =& \Gamma_\theta[\theta_k + \eta_\theta(k) \bigtriangledown_\theta L(\theta_k, \lambda_k)] \\ \bigtriangledown_\theta L(\theta_k, \lambda_k) =& -\mathbb{E}_{s\sim p, a\sim\pi_{\theta_k}}[C(s,a) - d] \\ \bigtriangledown_\lambda L(\theta_k, \lambda_k) =& \mathbb{E}_{s\sim p, a\sim\pi_{\theta_k}}[\bigtriangledown_{\theta_k} log\pi_{\theta_k}(a|s)(Q(s,a) - \lambda C(s,a))] \end{aligned} \tag{3.2}$$

where Γ_λ and Γ_θ are project operators. Γ_λ projects the λ into a feasible region $[0, \lambda_{max}]$ and Γ_θ projects the θ onto a compact and convex set to make θ stable.

However, unlike the long-term discounted return, which is approximated by Critic, there is no approximation function for the cost constraint. Therefore, the Critic is redefined in RCPO as a penalized reward function as follows:

$$\begin{aligned} \hat{r}(\lambda, s, a) &= r(s,a) - \lambda c(s.a) \\ \hat{V}^\pi(\lambda, s) &= \mathbb{E}_{s\sim p, a\sim\pi}[\sum_{t=0}^{\infty} \hat{r}(\gamma\lambda, s, a)|s_0 = s] \end{aligned} \tag{3.3}$$

3.1.2 Constrained Policy Optimization

The Constrained Policy Optimization (CPO) algorithm [24] is a safe reinforcement learning approach that builds upon the Trust Region Policy Optimization (TRPO) method. In conventional policy search algorithms, the policy is iteratively updated by maximizing the reward function $J(\pi)$ while being constrained within a local neighborhood of the previous policy π_k. For instance, in TRPO, the policy update rule can be expressed as follows, where π_θ represents the policy π parameterized by θ:

$$\begin{aligned} &\theta_{k+1} = \arg\max_\theta \mathbb{L}(\theta_k, \theta) \\ &s.t. \quad \bar{D}_{KL}(\theta||\theta_k) \leq \delta \end{aligned} \tag{3.4}$$

The term $\mathbb{L}$ represents the surrogate advantage function. It's able to evaluate the performance of the policy π_θ in comparison to the old policy π_{θ_k}. It is computed using the trajectory obtained from the execution of the old policy.

$$\mathbb{L}(\theta_k, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)] \tag{3.5}$$

$\bar{D}_{KL}(\theta||\theta_k)$ is a distance measure called average KL-divergence that measures the distance between two probability distributions. δ represents the distance threshold, which also can be seen as the step size.

$$\begin{aligned} \bar{D}_{KL}(\theta||\theta_k) &= \mathbb{E}_{s\sim\pi_{\theta_k}}[D_{KL}(\pi_\theta(.|s)||\pi_{\theta_k}(.|s))] \\ &= \mathbb{E}_{s\sim\pi_{\theta_k}}(\pi_\theta(.|s) log \frac{\pi_\theta(.|s)}{\pi_{\theta_k}(.|s)}) \end{aligned} \tag{3.6}$$

This distance constraint ensures the policy change within a small region called the trust region. The motivation for using the trust region method can be listed as follow:

- Guaranteed monotonic improvement: By using the trust regions method, the policy is updated only within a region where the performance is guaranteed to improve. This leads to monotonic improvement of policy performance, which is a desirable property in reinforcement learning. Neural network policies, in particular, are susceptible to performance collapse following unfavorable updates.
- Improved stability: Due to the limited step size, TRPO is less likely to overreact to small fluctuations in the reward signal and diverge. This helps to reduce the impact of noise in the environment or in the learning process.
- Improved sample efficiency: Since the policy updates are small and stable, the risk of the policy getting stuck in suboptimal regions of the policy space is reduced. This allows TRPO to converge to a good policy faster and with fewer samples than other reinforcement learning algorithms.

Based on the above definitions, the update rule for CPO is as follows:

$$\begin{aligned} &\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s\sim\pi_{\theta_k}, a\sim\pi_\theta}[A^{\pi_{\theta_k}}(s,a)] \\ &s.t. \quad J_{Ci}(\pi_{\theta_k}) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim\pi_{\theta_k}, a\sim\pi_\theta}[A_{Ci}^{\pi_{\theta_k}}(s,a)] \le d_i \quad v_i = 1, 2...m \\ &s.t. \quad \bar{D}_{KL}(\theta||\theta_k) \le \delta \end{aligned} \tag{3.7}$$

However, solving Equation 3.7 directly can be impractical for policies with high-dimensional parameter spaces, such as those represented by neural networks. This is due to the significant computational cost associated with evaluating and optimizing these complex models. But as stated in the trust region method, the update step size is strictly limited within a small range. By leveraging the property of local linearity around the last policy π_θ, both the reward and cost functions can be effectively approximated. Additionally, the KL divergence constraint can be approximated using a Taylor expansion. As a result, the problem can be transformed into a more manageable and simplified form.

$$\begin{aligned} \theta_{k+1} &= \arg\max_{\theta} g^T(\theta - \theta_k) \\ s.t. \quad & c_i + b_i^T(\theta - \theta_k) \le 0 \quad i = 1, 2...m \\ & c_i = J_{Ci}(\pi_{\theta_k}) - d_i \\ s.t. \quad & \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \le \delta \end{aligned} \tag{3.8}$$

In the given statement, g represents the gradient of the objective function, b_i is the gradient of the cost function, and H represents the Hessian matrix of the Kullback–Leibler divergence. By this means, the complex CMDP problem is further transferred into a relatively simpler convex problem and can be solved efficiently using the duality approach. Especially for a problem with only one constraint, it's able to calculate an analytical solution.

3.2 Stackelberg model and bi-level reinforcement learning

3.2.1 Stackelberg model

Stackelberg model was first introduced by Heinrich von Stackelberg, a German economist, in his book [25]. Nowadays, the Stackelberg model is often used as a game theory concept to model duopolies, which means the business competition between two companies. The unique property that differs the Stackelberg model from other commonly used models such as Cournot and Bertrand is that the players don't decide their actions simultaneously but sequentially. One player called the leader is allowed to commit an action first and the other player called the follower can choose his action based on the leader's action. Another important assumption made in this model is that each player knows or can infer how the opponents will react. As for the leader, since it has to move first, it should choose an action that can maximize its payoff considering the follower's response to his action. After the leader chooses his action, it must commit to it and is not allowed to change. As for the follower, it is therefore forced to react to the action chosen by the leader to maximize its own payoff. After both players have decided on their strategies, their actions will be executed simultaneously. The solution achieved by solving this Stackelberg model is called the Stackelberg equilibrium. To illustrate this model, an example of competition between two firms is given as follows:

game settings

- Firm 1 choose a quantity of production $q_1 \ge 0$ and announces its quantity.

- Firm 2 sees Firm 1′s quantity and then chooses a quantity of production of $q_2 \geq 0$
- the market sets the price of the production
  - denote $q = q_1 + q_2$ as the total quantity of production
  - price function $p(q) = a - q$ outputs the price p for $a \geq q$, where a is a constant
  - for $a < q$, $p(q) = 0$
- Firms have margin cost of production $c_i \geq 0$
- Objective function(profit): $p(q)q_i - c_i q_i$

Firm 2′s best response

To solve this Stackelberg game, Firm 1 has first to solve the best response for Firm 2 given Firm 1′s quantity. The profit of Firm 2 can be calculated as follow:

$$\begin{aligned} f_2(q_2) =& (a - q_1 - q_2)q_2 - c_2 q_2 \\ =& a q_2 - q_1 q_2 - q_2^2 - c_2 q_2 \end{aligned} \tag{3.9}$$

use the first-order condition to calculate the maximum point of q_2.

$$\begin{aligned} \bigtriangledown_{q_2} f_2(q_2) =& a - q_1 - 2q_2 - c_2 = 0 \\ q_2 =& (a - q_1 - c_2)/2 \end{aligned} \tag{3.10}$$

Firm 1′s optimization problem

Since now the best response function of Firm 2 is available, the quantity q_2 in Firm 1′s profit function can be substituted with the best response function.

$$\begin{aligned} f_1(q_1) =& (a - q_1 - q_2)q_1 - c_1 q_1 \\ =& (a q_1 - q_1^2 + c_2 q_1)/2 - c_1 q_1 \end{aligned} \tag{3.11}$$

Then Firm 1 can choose a quantity that maximizes this profit function.

$$\begin{aligned} \bigtriangledown_{q_1} f_1(q_1) =& (a - 2q_1 + c_2)/2 - c_1 = 0 \\ q_1 =& (a + c_2)/2 - c_1 \end{aligned} \tag{3.12}$$

Equilibrium

Input Firm 1′s solution into Firm 2′s best response function, and we can obtain the Stackelberg Equilibrium of this game.

$$\begin{aligned} q_1 &= (a + c_2)/2 - c_1 \\ q_2 &= a/4 - 3c_2/4 + c_1/2 \end{aligned} \tag{3.13}$$

3.2.2 Bi-level Actor Critic

Many of the current multi-agent reinforcement learning algorithms approach cooperative games with a symmetric perspective, treating each agent equally. However, this approach often leads to arbitrary Nash equilibria in Markov games, particularly when multiple equilibria exist. The behavior learned by each agent may suffer from uncertainty and sub-optimality. In their paper [3], the authors address this challenge by adopting an asymmetric Stackelberg model. The choice of Stackelberg equilibrium over Nash equilibrium is motivated by its potential as a convergence point that offers superior Pareto optimality, particularly in cooperative environments.

base on the Bellman equation and the Stackelberg model the update rule for the bilevel Q-learning can be formulated as follows:

$$\begin{aligned} & a'_1 \leftarrow \arg\max_{a_1} Q_1(s', a_1, \arg\max_{a_2} Q_2(s', a_1, a_1)) \\ & a'_2 \leftarrow \arg\max_{a_2} Q_2(s', a'_1, a_2) \\ & Q_1(s, a_1, a_2) \leftarrow (1 - \alpha_1) Q_1(s, a_1, a_2) + \alpha_1 (r_1 + \gamma Q_1(s', a'_1, a'_2)) \\ & Q_2(s, a_1, a_2) \leftarrow (1 - \alpha_2) Q_2(s, a_1, a_2) + \alpha_2 (r_2 + \gamma Q_2(s', a'_1, a'_2)) \end{aligned} \tag{3.14}$$

Where Agent 1 is the leader and Agent 2 is the follower. However, the applicability of this algorithm is currently limited, as it can only be effectively employed in scenarios with simple discrete action and state. Once applied to a complex discrete or continuous action space, the computational load of the bilevel optimization problem during iteration will increase significantly. To address this challenge, the researchers propose a solution called the bi-level Actor-Critic (Bi-AC) method. In this approach, the policy actor is introduced to represent the follower agent, while the leader agent is maintained as a Q-learner. Agent 2′s(follower) policy(actor) is denoted by a neural network $\pi_2(s, a_1)$ with parameter ϕ_2, which takes the current state and agent 1′s(leader) action as input. Agent 2′s actor can be the response function that reacts to Agent 1′s action. Two Q-value functions(critic) are also approximated as neural networks $Q_i(s, a_1, a_2)$ with parameter θ_i that takes the current state and all agents′ actions as input. The update rule is given as follows:

$$\begin{aligned}
&a'_1 \leftarrow \arg\max_{a_1} Q_1(s', a_1, \pi_2(s', a_1)) \\
&a'_2 \leftarrow \pi_2(s', a'_1) \\
&y_i \leftarrow r_i + \gamma Q_i(s', a'_1, a'_2) - Q_i(s, a_1, a_2), i = 1, 2 \\
&\theta_i \leftarrow \theta_i - \alpha_i \bigtriangledown_{\theta_i} y_i^2, i = 1, 2 \\
&\phi_2 \leftarrow \phi_2 + \beta \bigtriangledown_{\phi_2} log \pi_2(s, a_1) Q_2(s, a_1, a_2)
\end{aligned} \tag{3.15}$$

α_i and β_i are the learning rate. The parameter update is similar to the MADDPG algorithm.

The Bi-AC algorithm is able to be further extended to accommodate multi-level architectures by training an actor and a critic for each agent involved in the multi-agent system. For continuous action space, the actor can be defined as follow:

$$\mu_i(s, a_1, a_2 ... a_{i-1}), i = 1, 2 ... n \tag{3.16}$$

The actor inputs the current state and all upper-level agents' actions. The critic is similar to two agents scenario:

$$Q_i(s, a_1, a_2 ... a_n), i = 1, 2 ... n \tag{3.17}$$

To prove the convergence of the bi-level actor-critic algorithm. Matrix games and autonomous driving experiments are conducted on this algorithm. It shows the ability of successful convergence to the Stackelberg Equilibrium and finding an asymmetric solution.

4 Constrained Stackelberg Multi-Agent Reinforcement Learning

In this chapter, our emphasis lies on the utilization of the constrained Stackelberg model in the context of multi-agent reinforcement learning domains. We will explore how the constrained Stackelberg model can be integrated with Q-learning and MADDPG algorithms. We will discuss the formulation of the model, the adaptation of Q-learning and MADDPG to incorporate the Stackelberg dynamics and the methods to handle the constraints.

4.1 Problem Formulation

Let's start with unconstrained Stackelberg reinforcement learning and consider a two-agent system consisting of a leader and follower agents. In this problem, we consider a leader-follower scenario where the leader agent aims to optimize its policy to maximize its total return, considering the anticipated response of the follower agent. On the other hand, the follower agent observes the leader's policy and selects its own optimal policy accordingly. Thus, we can view this reinforcement learning problem as a bi-level optimization problem.

$$\begin{aligned} \max_{\pi_1} \quad & J_1(\pi_1, \pi_2^*) = \mathbb{E}_{a_1,a_2\sim\pi_1,\pi_2^*} \sum_{t=1}^{\infty} \gamma^t r_1^t \\ \text{s.t.} \quad & \pi_2^* = \max_{\pi_2} J_2(\pi_1, \pi_2) = \mathbb{E}_{a_1,a_2\sim\pi_1,\pi_2} \sum_{t=1}^{\infty} \gamma^t r_2^t \end{aligned} \tag{4.1}$$

where J_i denotes the expected cumulative reward of agent i if agents adhere to their policies π_1 and π_2. This problem can be referred to as bi-level reinforcement learning (BiRL), which can be seen as an extension of the Stackelberg model in a multi-state setting[25].

To incorporate constraints in the Stackelberg reinforcement learning framework, we introduce a set of constraints that restrict the agents' actions or policies based on the constrained Markov decision process. These constraints can be defined based on safety requirements, resource limitations, or other system-specific constraints. In addition to optimizing the objective, one or more constraints have to be controlled within a threshold by the agents. In our problem, the constraint is measured as the long-term accumulated discounted costs by following π_1 and π_2.

$$G_{i,j}(\pi_1, \pi_2) = \mathbb{E}_{a_1,a_2 \sim \pi_1,\pi_2} \sum_{t=1}^{\infty} \gamma^t c_{i,j}^t \quad j = 1, 2...m \tag{4.2}$$

where $G_{i,j}$ denotes the jth constraint of Agent i and $c_{i,j}^t$ denotes the jth cost of Agent i at time step t. By augmenting problem 4.1 with constraints, the constrained Stackelberg multi-agent reinforcement learning problem can be formulated as follow:

$$\begin{aligned}
\max_{\pi_1} \quad & J_1(\pi_1, \pi_2^*) = \mathbb{E}_{a_1,a_2 \sim \pi_1,\pi_2^*} \sum_{t=1}^{\infty} \gamma^t r_1^t \\
\text{s.t.} \quad & G_{1,j}(\pi_1, \pi_2^*) = \mathbb{E}_{a_1,a_2 \sim \pi_1,\pi_2^*} \sum_{t=1}^{\infty} \gamma^t c_{1,j}^t \le d_{1,j} \quad \forall j = 1...m_1 \\
& \pi_2^* = \max_{\pi_2} \quad J_2(\pi_1, \pi_2) = \mathbb{E}_{a_1,a_2 \sim \pi_1,\pi_2} \sum_{t=1}^{\infty} \gamma^t r_2^t \\
& \text{s.t.} \quad G_{2,j}(\pi_1, \pi_2) = \mathbb{E}_{a_1,a_2 \sim \pi_1,\pi_2} \sum_{t=1}^{\infty} \gamma^t c_{2,j}^t \le d_{2,j} \quad \forall j = 1...m_2
\end{aligned} \tag{4.3}$$

The constraints can be formulated as inequalities or equalities depending on the specific requirements. Here we use the inequality constraints, where $d_{i,j}$ represents the threshold limit for the jth constraint of Agent i. This problem can, therefore, also be called constrained bi-level reinforcement learning. Several tools can be leveraged to address the constrained bi-level optimization problem, e.g., trust-region methods [26] and branch-and-bound methods [27]. However, for complex reinforcement learning problems, especially those with continuous state and action spaces, the reward and constraint functions are unknown and can only be approximated using neural networks. And as the number of dimensions increases, the difficulty of problem-solving also exponentially rises.

4.2 Constrained Stackelberg Q-learning

Directly solving the above problem in 4.3 is generally impractical. We start by solving a simple case with discrete action space and continuous state space using the Q-learning method. For a simple action space, the bi-level optimization problem can be solved by enumerating all action combinations. The following question only considers cases where each agent has only one constraint.

4.2.1 Value functions for reward and cost

In the context of constrained Q-learning, the value function represents the expected accumulated reward that an agent can achieve by following a particular policy in a given state. However, in addition to the reward, constrained Q-learning also considers a cost-value function associated with actions and state transitions to enforce constraints. The Q-functions for reward and cost are defined as follows and approximated using neural networks. Here we use a centralized version of the Q-function that takes a global state or observation and all agents' actions as input. Each agent has to maintain its Q-function networks.

$$Q_i(s, a_1, a_2) = \mathbb{E}_{s' \sim P}(r_i(s, a_1, a_2) + \gamma \mathbb{E}_{a_1', a_2' \sim \pi_1, \pi_2} Q_i(s', a_1', a_2')) \tag{4.4}$$

$$G_i(s, a_1, a_2) = \mathbb{E}_{s' \sim P}(c_i(s, a_1, a_2) + \gamma \mathbb{E}_{a_1', a_2' \sim \pi_1, \pi_2} G_i(s', a_1', a_2')) \tag{4.5}$$

Q_i stands for the Q-function for reward with parameter ϕ_i and G_i for cost with parameter ζ_i. P denotes the state transition distribution when following π_1 and π_2. The ideal policy here is to adhere to the equilibrium point of the constrained Stackelberg model, which will be described below.

4.2.2 Bellman equation

Just like the min-max-Q and Nash-Q approaches, we can define the constrained Stackelberg-Bellman equation for reward and cost in the following manner:

$$Q_i^{t+1}(s, a_1, a_2) = (1 - \alpha) Q_i^t(s, a_1, a_2) + \alpha(r_i^t + \gamma c\_StackelbergQ_i^t(s')) \tag{4.6}$$

$$G_i^{t+1}(s, a_1, a_2) = (1 - \alpha) G_i^t(s, a_1, a_2) + \alpha(c_i^t + \gamma c\_StackelbergG_i^t(s')) \tag{4.7}$$

$$c\_StackelbergQ_i^t(s') = \pi_1(s') \pi_2(s') Q_i^t(s') \tag{4.8}$$

$$c\_StackelbergG_i^t(s') = \pi_1(s') \pi_2(s') G_i^t(s') \tag{4.9}$$

where $c\_StackelbergQ_i^t(s')$ and $c\_StackelbergG_i^t(s')$ denotes the i-th agent's expected reward and cost in the constrained Stackerberg equilibrium point for state s. Under the guidance of this Bellman equation, we can update the Q-functions iteratively.

4.2.3 Update rule

With a transaction $\langle s, a_1, a_2, r_1, r_2, c_1, c_2, d, s' \rangle$, we need to first find the Stackelberg equilibrium point of 4.10 for the next state s' by enumerating all action combinations.

$$\begin{aligned} a'_1 &\leftarrow \arg\max_{a_1} Q_1(s', a_1, \arg\max_{a_2 \in A_2(s')} Q_2(s', a_1, a_2)) \\ &\text{s.t.} \quad G_1(s', a_1, a_2) \le d_1 \\ a'_2 &\leftarrow \arg\max_{a_2} Q_2(s', a'_1, a_2) \\ &\text{s.t.} \quad G_2(s', a'_1, a_2) \le d_2 \end{aligned} \tag{4.10}$$

$A_2(s')$ is the safe set of Agent 2's action in state s' and d_i represents the threshold of Agent i's constraint. The parameters ϕ_i and ζ_i are then updated using gradient descent to minimize the TD error of the corresponding Q-functions.

$$\begin{aligned} \phi_1 &\leftarrow \phi_1 - \alpha_1 \bigtriangledown_{\phi_1} (Q_1(s, a_1, a_2) - r_1 - \gamma(1-d) Q_1^{targ}(s', a'_1, a'_2)) \\ \phi_2 &\leftarrow \phi_2 - \alpha_2 \bigtriangledown_{\phi_2} (Q_2(s, a_1, a_2) - r_2 - \gamma(1-d) Q_2^{targ}(s', a'_1, a'_2)) \\ \zeta_1 &\leftarrow \zeta_1 - \beta_1 \bigtriangledown_{\zeta_1} (G_1(s, a_1, a_2) - c_1 - \gamma(1-d) G_1^{targ}(s', a'_1, a'_2)) \\ \zeta_2 &\leftarrow \zeta_2 - \beta_2 \bigtriangledown_{\zeta_2} (G_2(s, a_1, a_2) - c_2 - \gamma(1-d) G_2^{targ}(s', a'_1, a'_2)) \end{aligned} \tag{4.11}$$

where α_i and β_i are the learning rate.

Algorithm 1 Constrained Stackelberg Q-learning(C_SQ)

```
Initialization
Initialize Q-function parameters $\phi_i$, cost value function parameters $\zeta_i$ and replay buffer $D$;
Initialize target networks and they share the same parameters as main networks $\phi_i^{targ} = \phi_i$, $\zeta_i^{targ} = \zeta$, $i \in \{1,2\}$;
Initialize hyperparameters $d_i$, $\gamma$ and $\rho$ $\phi_i^{targ} = \phi_i$, $\zeta_i^{targ} = \zeta$, $i \in \{1,2\}$;
repeat
  sample actions with $\epsilon$-greedy according to 4.10;
  $a_1^* \leftarrow \arg\max_{a_1} Q_1(s, a_1, \arg\max_{a_2} Q_2(s, a_1, a_2))$
    s.t. $G_1(s, a_1, a_2) \leq d_1$
  $a_2^* \leftarrow \arg\max_{a_2} Q_2(s, a_1^*, a_2)$
    s.t. $G_2(s, a_1^*, a_2) \leq d_2$
  observe next state $s'$, reward $r_i$, cost $c_i$, and done signal $d$ which indicates whether $s'$ is terminal;
  Store trajectory $(s, a_i, r_i, c_i, d, s')$ into replay buffer $D$;
  if $s'$ is terminal, reset environment state;
  if it's time to update then
    for however many updates do
      Randomly sample a batch of transitions, $B = \{(s, a_i, r_i, c_i, d, s')\}$ from $D$;
      compute target actions according to 4.10;
      compute target Q:
      $y_i = r_i + \gamma(1-d)Q_i^{targ}(s', a_1', a_2')$
      compute target cost:
      $g_i = c_i + \gamma(1-d)G_i^{targ}(s', a_1', a_2')$
      update critic function:
      $\nabla_\phi \frac{1}{|B|} \sum_{\{(s,a,r,c,s',d)\in R\}} (Q_i(s, a_1, a_2) - y_i)^2$
      update critic cost function:
      $\nabla_\zeta \frac{1}{|B|} \sum_{\{(s,a,r,c,s',d)\in R\}} (G_i(s, a_1, a_2) - g_i)^2$
      update target network:
      $\phi_i^{targ} \leftarrow \rho\phi_i^{targ} + (1-\rho)\phi_i$
      $\zeta_i^{targ} \leftarrow \rho\zeta_i^{targ} + (1-\rho)\zeta_i$
    end for
  end if
until converge
```

4.3 Constrained Stackelberg MADDPG

The application of the constrained Stackelberg Q-learning is strictly limited to scenarios with discrete action spaces, and as the action number goes on, the computation time will grow exponentially. To solve this problem, we propose the constrained Stackelberg MADDPG(C_SMADDPG) algorithm by adapting the MADDPG algorithm to the constrained Stackelberg model.

4.3.1 Actor, Critic and Cost Critic

First, let's recap some basic settings and definitions in MADDPG. MADDPG utilizes a critic-actor architecture for each agent. Given all agents ' joint actions and observations, the critic learns to evaluate the expected return. The actor learns to select actions based on the agent's local observations.

To align MADDPG with the Stackelberg model, certain redefinitions are required. In accordance with the Stackelberg setting, we designate Agent 1 as the leader and Agent 2 as the follower. This adjustment allows us to establish the hierarchical relationship between the two agents and better capture the strategic decision-making dynamics inherent in the Stackelberg framework. The leader agent's actor operates based on its local observations, independently determining its decision. Subsequently, the follower agent observes the decision made by the leader and then makes its own decision accordingly. Their deterministic policies are approximated by neural networks μ_i with parameter θ_i(stochastic policies are denoted as π_i). The hierarchical decision-making sequence is defined as follows.

$$\begin{aligned} a_1 &= \mu_1(o_1|\theta_1) \\ a_2 &= \mu_2(o_2, a_1|\theta 2) \end{aligned} \tag{4.12}$$

where o_i denotes the local observation of each agent. The decisions are therefore made sequentially but executed simultaneously and decentralized.

The critic's definition remains unchanged and refers to the centralized Q value function that estimates the expected accumulated reward of taking a set of specific joint actions in a given state. This value function is used to guide the agent towards actions that are likely to result in higher rewards. In our multi-agent reinforcement learning context, the critic network of Agent i with parameter ϕ_i can be denoted as follow(ϕ_i will be omitted afterward to minimize notation):

$$Q_i(s, a_1, a_2|\phi_i) \tag{4.13}$$

where s represents the global state or the set of observations of all agents, depending on the current environment setup.

In addition to the actor and critic, we must introduce a new cost critic(or penalty critic). The cost critic refers to a centralized value function that estimates the expected accumulated cost or penalty associated with taking a set of specific joint actions in a given state, similar to the critic. This value function is employed to steer the agent away from actions that are anticipated to incur higher costs or penalties. In this context, the cost critic network for the jth constraint of agent i with parameter $\zeta_{i,j}$ can be denoted as follow($\zeta_{i,j}$ will also be omitted afterward), where m_i is the number of agent i's constraints:

$$G_{i,j}(s, a_1, a_2|\zeta_{i,j}) \quad j = 1...m_i \tag{4.14}$$

4.3.2 Problem reformulation and Lagrangian Approach

To provide a more straightforward presentation of the specific problems we need to address, based on the aforementioned new definition, we can rewrite the constrained bi-level optimization problem 4.3 in two steps. Assuming that we already have well-approximated value and cost functions, the decision-making process at each time step is to solve the following constrained bi-level problem to get a set of optimal joint actions.

$$\begin{aligned} a_1^* &= \arg\max_{a_1} Q_1(s, a_1, a_2^*) \\ &\quad \text{s.t.} \quad G_{1,j}(s, a_1, a_2^*) \leq d_{1,j} \quad \forall j = 1...m_1 \\ a_2^* &= \arg\max_{a_2} Q_2(s, a_1, a_2) \\ &\quad \text{s.t.} \quad G_{2,j}(s, a_1, a_2) \leq d_{2,j} \quad \forall j = 1...m_2 \end{aligned} \tag{4.15}$$

However, solving such an optimization problem is impractical as it requires tremendous computational effort. That's why we need to use the actor. The actor only needs to perform a simple forward propagation to obtain actions. To achieve this, we need to incorporate the actors' parameters into the problem 4.15 and transform it into an optimization problem with respect to the parameter θ_i, shown below.

$$\begin{aligned} \theta_1^* &= \arg\max_{\theta_1} Q_1(s, \mu_1(o_1|\theta_1), \mu_2(o_2, \mu_1(o_1|\theta_1)|\theta_2^*)) \\ &\quad \text{s.t.} \quad G_{1,j}(s, \mu_1(o_1|\theta_1), \mu_2(o_2, \mu_1(o_1|\theta_1)|\theta_2^*) \leq d_{1,j} \quad \forall j = 1...m_1 \\ \theta_2^* &= \arg\max_{\theta_2} Q_2(s, \mu_1(o_1|\theta_1), \mu_2(o_2, \mu_1(o_1|\theta_1)|\theta_2) \\ &\quad \text{s.t.} \quad G_{2,j}(s, \mu_1(o_1|\theta_1), \mu_2(o_2, \mu_1(o_1|\theta_1)|\theta_2) \leq d_{2,j} \quad \forall j = 1...m_2 \end{aligned} \tag{4.16}$$

To solve 4.16, we employ a Lagrangian relaxation approach, which leads to the unconstrained bi-level problem:

$$\begin{aligned}\max_{\theta_1}\min_{\lambda_1\geq 0} L_1(\theta_1,\lambda_1) =& Q_1(s,\mu_1(o_1|\theta_1),\mu_2(o_2,\mu_1(o_1|\theta_1)|\theta_2^*))- \\ &\sum_{j=0}^{m_1}\lambda_{1,j}(G_{1,j}(s,\mu_1(o_1|\theta_1),\mu_2(o_2,\mu_1(o_1|\theta_1)|\theta_2^*)-d_{1,j}) \\ \text{s.t.} \max_{\theta_2}\min_{\lambda_2\geq 0} L_2(\theta_2,\lambda_2) =& Q_2(s,\mu_1(o_1|\theta_1),\mu_2(o_2,\mu_1(o_1|\theta_1)|\theta_2))- \\ &\sum_{j=0}^{m_2}\lambda_{2,j}(G_{2,j}(s,\mu_1(o_1|\theta_1),\mu_2(o_2,\mu_1(o_1|\theta_1)|\theta_2)-d_{2,j})\end{aligned} \tag{4.17}$$

where $\lambda_{i,j}$ denotes the Lagrange multipliers. This implies the adoption of a two-timescale approach: at a faster timescale, the optimal values of θ are determined by solving equation 4.17, whereas at a slower timescale, the Lagrange multiplier λ is gradually increased until the constraints are successfully met.

4.3.3 Actor update

In our reinforcement learning environment, we only consider two agents, and each agent has a constraint number of 1, i.e., $m_1 = 1, m_2 = 1$. With the guidance of the above equations, we can derive the update rule for the actors.

$$\begin{aligned}\theta_1 &\leftarrow \theta_1 + \alpha_1 \bigtriangledown_{\theta_1} L_1(\theta_1,\lambda_1) \\ \lambda_1 &\leftarrow \lambda_1 - \beta_1 \bigtriangledown_{\lambda_1} L_1(\theta_1,\lambda_1) \\ \theta_2 &\leftarrow \theta_2 + \alpha_2 \bigtriangledown_{\theta_2} L_2(\theta_2,\lambda_2) \\ \lambda_2 &\leftarrow \lambda_2 - \beta_2 \bigtriangledown_{\lambda_2} L_2(\theta_2,\lambda_2)\end{aligned} \tag{4.18}$$

where α_i and β_i denote the learning rate of θ_i and λ_i. The derivatives in (3.17) with respect to θ and λ can be computed with a set of trajectory samples D that are randomly sampled from the replay buffer. We can obtain the following actor update rules according to the chain rule.

Leader

$$\begin{aligned}\bigtriangledown_{\theta_1} L_1(\theta_1,\lambda_1) =& \mathbb{E}_{\tau\sim D} \bigtriangledown_{\theta_1} \mu_1(o_1)(\bigtriangledown_{a_1} Q_1(s,a_1,a_2) + \bigtriangledown_{a_1}\mu_2(o_2,a_1) \bigtriangledown_{a_2} Q_1(s,a_1,a_2)- \\ &\sum_{j=0}^{m_1}\lambda_{1,j} \bigtriangledown_{a_1} G_{1,j}(s,a_1,a_2) - \bigtriangledown_{a_1}\mu_2(o_2,a_1) \bigtriangledown_{a_2} G_{1,j}(s,a_1,a_2))\end{aligned} \tag{4.19}$$

$$\nabla_{\lambda_1} L_1(\theta_1, \lambda_1) = -\mathbb{E}_{\tau \sim D} \sum_{j=0}^{m_1} \lambda_{1,j}(G_{1,j}(s, a_1, a_2)) - d_{1,j}) \tag{4.20}$$

Follower

$$\nabla_{\theta_2} L_2(\theta_2, \lambda_2) = \mathbb{E}_{\tau \sim D} \nabla_{\theta_2} \mu_2(o_2, a_1)(\nabla_{a_2} Q_2(s, a_1, a_2) - \sum_{j=0}^{m_2} \lambda_{2,j} \nabla_{a_2} G_{2,j}(s, a_1, a_2)) \tag{4.21}$$

$$\nabla_{\lambda_2} L_2(\theta_2, \lambda_2) = -\mathbb{E}_{\tau \sim D} \sum_{j=0}^{m_2} \lambda_{2,j}(G_{2,j}(s, a_1, a_2)) - d_{2,j}) \tag{4.22}$$

4.3.4 Critic and Cost Critic update

The centralized value function and cost value function Q_i and $G_{i,j}$ are updated using off-policy temporal difference learning as defined in the MADDPG section. The update rule is identical to that of MADDPG but with a hierarchical decision-making process. Their networks are updated by minimizing the temporal difference error using gradient descent:

Critic

$$\begin{aligned} L(\phi_i) =& \mathbb{E}_{\tau \sim D}(Q_i(s, a_1, a_2) - y)^2 \\ y =& r_i + (1-d)\gamma Q_i^{targ}(s', a'_1, a'_2) \\ a'_1 =& \mu_1^{targ}(o'_1) \\ a'_2 =& \mu_2^{targ}(o'_2, a'_1) \end{aligned} \tag{4.23}$$

Cost Critic

$$\begin{aligned} L(\zeta_{i,j}) =& \mathbb{E}_{\tau \sim D}(G_{i,j}(s, a_1, a_2) - y)^2 \\ y =& c_i + (1-d)\gamma G_{i,j}^{targ}(s', a'_1, a'_2) \\ a'_1 =& \mu_1^{targ}(o'_1) \\ a'_2 =& \mu_2^{targ}(o'_2, a'_1) \end{aligned} \tag{4.24}$$

Where μ_i^{targ} is the target policy with delayed parameters θ_i^{targ}, and Q_i^{targ}, $G_{i,j}^{targ}$ are the target critics, and target cost critics with delayed parameters ϕ_i^{targ} and $\zeta_{i,j}^{targ}$. These target networks are updated periodically with the most recent parameters, which helps stabilize learning.

Algorithm 2 Constrained Stackelberg MADDPG(C_SMADDPG)

```
**Initialization**
Initialize policy parameters $\theta_i$, Q-function parameters $\phi_i$, cost value function parameters $\zeta_i$ and empty replay buffer $D$;
Initialize target networks and they share the same parameters as main networks $\theta_i^{targ} = \theta_i$, $\phi_i^{targ} = \phi_i$, $\zeta_i^{targ} = \zeta_i$, $i \in \{1,2\}$;
Initialize hyperparameters $d_i$, $\gamma$, $\alpha_i$, $\beta_i$ and $\rho$ , $i \in \{1,2\}$;
**repeat**
  Observe state s and select action:
    $a_1 = clip(\mu_1(s) + \epsilon, a_{low}, a_{high})$,
    $a_2 = clip(\mu_2(s, a_1) + \epsilon, a_{low}, a_{high})$, where $\epsilon \sim \mathbb{N}$
  execute $(a_1, a_2)$ in the environment;
  observe next state $s'$, reward $r_i$, cost $c_i$, and done signal $d$ which indicates whether $s'$ is terminal;
  Store trajectory $(s, a_i, r_i, c_i, d, s')$ into replay buffer $D$;
  if $s'$ is terminal, reset environment state;
  **if** it's time to update **then**
    **for** however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a_i, r_i, c_i, d, s')\}$ from $D$;
      compute target actions:
      $a_1' = \mu_1(s)$
      $a_2' = \mu_2(s, a_1')$
      compute target Q:
      $y_i = r_i + \gamma(1-d)Q_i^{targ}(s', a_1', a_2')$
      compute target cost:
      $g_i = c_i + \gamma(1-d)G_i^{targ}(s', a_1', a_2')$
      update critic function:
      $\nabla_\phi \frac{1}{|B|} \sum_{\{(s,a,r,c,s',d) \in R\}} (Q_i(s, a_1, a_2) - y_i)^2$
      update critic cost function:
      $\nabla_\zeta \frac{1}{|B|} \sum_{\{(s,a,r,c,s',d) \in R\}} (G_i(s, a_1, a_2) - g_i)^2$
      update leader and follower's actor using policy gradient 4.3.3:
      $\theta_i^{k+1} = \theta_i^k + \alpha_i \bigtriangledown_{\theta_i} L_i(\theta_i, \lambda_i)$
      update leader and follower's Lagrange multiplier 4.3.3:
      $\lambda_i^{k+1} = \lambda_i^k + \beta_i \bigtriangledown_\lambda L_i(\theta_i, \lambda_i)$
      update target network:
      $\theta_i^{targ} \leftarrow \rho\theta_i^{targ} + (1-\rho)\theta_i$
      $\phi_i^{targ} \leftarrow \rho\phi_i^{targ} + (1-\rho)\phi_i$
      $\zeta_i^{targ} \leftarrow \rho\zeta_i^{targ} + (1-\rho)\zeta_i$
    **end for**
  **end if**
**until** converge
```

5 Convergence

We aim to examine how Q_i^t for each learning Agent i converges to a constant equilibrium Q_i^*. The optimal value function Q_i^* is established by the constrained Stackelberg strategies of leader and follower agents. By leveraging bi-level optimization techniques, we derive effective strategies based on these learned Q-values. In our study, we focus on two agents: the leader agent and the follower agent. We strive to achieve the optimal value function (Q_1^*, Q_2^*) as our learning objective and demonstrate the convergence of (Q_1, Q_2) to (Q_1^*, Q_2^*). Our convergence analysis is built on the proof of Nash-Q in [1] and Bi-level Actor-Critic in [3].

5.1 Convergence analysis of Constrained Stackelberg Q-learning algorithm

Assumption 1 *Every state s in the state space S and every action a_k in the action space A_k for agents $k = 1, 2$ are visited an infinite number of times.*

Assumption 2 *The learning rate α^t is defined to meet the following conditions for all s, t, a_1, and a_2:*
1. $0 \leq \alpha^t(s, a_1, a_2) < 1, \sum_{t=0}^{\infty} \alpha^t(s, a_1, a_2) = \infty, \sum_{t=0}^{\infty} [\alpha^t(s, a_1, a_2)]^2 < \infty$, and the latter two hold uniformly and with probability one.
2. $\alpha^t(s, a_1, a_2) = 0$ if $(s, a_1, a_2) \neq (s^t, a_1^t, a_2^t)$

Assumption 3 *For every state s in the state space S, the safe set of action space $A_k^{safe}(s)$ is not empty for agents $k = 1, 2$.*

Lemma 1 *(Szepesvari and Littman (1999), Corollary 5). Assume that α^t satisfies Assumption 2, and the mapping $P^t : \mathbb{Q} \to \mathbb{Q}$ satisfies the following condition: there exists a number $0 < \gamma < 1$ and a sequence $\lambda^t \geq 0$ converging into zero with probability one such that $||P^t Q - P^t Q^*|| \leq \gamma ||Q - Q^*|| + \lambda^t$ for all $Q \in \mathbb{Q}$ and $Q^* = E[P^t Q^*]$, then the iteration defined by*
$Q^{t+1} = (1 - \alpha)Q + \alpha^t [P^t Q^t]$ converges to Q^* with probability one.

Note that P_t in Lemma 1 is a pseudo-contraction operator because a "true" contraction operator should map every two points in the space closer to each other, which means $||P_t Q - P_t \hat{Q}|| \leq$

$\gamma||Q - \hat{Q}||$ for all $Q, \hat{Q} \in \mathbf{Q}$. Even when $\lambda_t = 0$, P_t is still not a contraction operator because it only maps every $Q \in \mathbf{Q}$ closer to Q^*, not mapping any two points in $\mathbf{Q}$ closer.

Definition 1 *Let $Q = (Q_1, Q_2)$ where $Q_1 \in \mathbb{Q}_1, Q_2 \in \mathbb{Q}_2$, and $\mathbb{Q} = \mathbb{Q}_1 \times \mathbb{Q}_2$, $P^t : \mathbb{Q} \to \mathbb{Q}$ is a mapping on the complete metric space $\mathbb{Q} \to \mathbb{Q}$, $P^tQ = (P^tQ_1, P^tQ_2)$, where*

$$P^tQ_i(s, a_1, a_2) = r_i^t(s, a_1, a_2) + \gamma c\_StackelbergQ_i^t(s'), for \quad i = 1, 2 \tag{5.1}$$

where s' is the state at time t+1, and $c\_StackelbergQ_i^t(s') = Q_i^t(s', \pi_1(s'), \pi_2(s'))$ is a constrained Stackelberg equilibrium solution for the game $(Q_1(s'), Q_2(s'))$ defined in section 3.2.2.

Lemma 2 *For an 2-player stochastic or deterministic game, $E[P^tQ^*] = Q^*$, where $Q^* = (Q_1^*, Q_2^*)$.*
$Q_i^*(s, a_1, a_2)$
$= r_i(s, a_1, a_2) + \gamma \sum p(s'|s, a_1, a_2) c\_StackelbergQ_i^*(s')$
$= r_i(s, a_1, a_2) + \gamma \sum p(s'|s, a_1, a_2) Q_i^*(s', \pi_1(s'), \pi_2(s'))$
$= \sum p(s'|s, a_1, a_2)(\gamma Q_i^*(s', \pi_1(s'), \pi_2(s')) + r_i(s, a_1, a_2))$
$= E[P_i^t Q_i^*(s, a1, a2)]$
i.e., $E[P^tQ^*] = Q^*$.

Assumption 4 *For every t and s, $(Q_1^t(s), Q_2^t(s))$ has a global optimal point, which is the constrained Stackelberg equilibrium point, and agents' payoffs at this optimal point are selected with 100% certainty since all possible combinations of actions have been explored.*

Definition 2 *The joint strategy (σ_1, σ_2) is the global optimal point of constrained Stackelberg game $(Q_1(s), Q_2(s))$.*
For the sake of simplicity in notation, we will use σ_i to denote $\pi_i(s)$ and $\hat{\sigma}_i$ to represent $\hat{\pi}_i(s)$.
For agent 1(leader agent), its payoff at the optimal point is higher than the payoff obtained by any other joint strategies:
$\sigma_1\sigma_2Q_1^j(s) \geq \hat{\sigma}_1\hat{\sigma}_2Q_1^j(s)$ for all $\hat{\sigma}_1, \hat{\sigma}_2 \in \sigma(A_1^{safe}(s)), \sigma(A_2^{safe}(s)))$
For agent 2(follower agent), its payoff at the optimal point is higher than the payoff obtained by any other follower strategies:
$\sigma_1\sigma_2Q_2^j(s) \geq \sigma_1\hat{\sigma}_2Q_2^j(s)$ for all $\hat{\sigma}_2 \in \sigma(A_2^{safe}(s))$

Definition 3 *For $Q, \hat{Q} \in \mathbf{Q}$, define:*

$$\begin{aligned} &||Q - \hat{Q}|| \\ &\equiv \max_j \max_s ||Q_j(s) - \hat{Q}_j(s)|| \\ &\equiv \max_j \max_s \max_{a_1, a_2} ||Q_j(s, a_1, a_2) - \hat{Q}_j(s, a_1, a_2)|| \end{aligned} \tag{5.2}$$

Lemma 3 $||P^t Q - P^t \hat{Q}|| \le \gamma ||Q - \hat{Q}||$ *for all* $Q \in \mathbf{Q}$

Proof:

$$\begin{aligned} ||P^t Q - P^t \hat{Q}|| &= \max_j ||P^t Q_j - P^t \hat{Q}_j|| \\ &= \max_j \max_s |\gamma \pi_1(s) \pi_2(s) Q_j(s) - \gamma \hat{\pi}_1(s) \hat{\pi}_2(s) \hat{Q}_j(s)| \\ &= \max_j \gamma |\pi_1(s) \pi_2(s) Q_j(s) - \hat{\pi}_1(s) \hat{\pi}_2(s) \hat{Q}_j(s)| \end{aligned} \tag{5.3}$$

We proceed to prove that

$$|\pi_1(s)\pi_2(s)Q_j(s) - \hat{\pi}_1(s)\hat{\pi}_2(s)\hat{Q}_j(s)| \le |Q_j(s) - \hat{Q}_j(s)| \tag{5.4}$$

After simplifying the notation:

$$|\sigma_1\sigma_2 Q_j(s) - \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_j(s)| \le |Q_j(s) - \hat{Q}_j(s)| \tag{5.5}$$

Leader agent(Agent 1)

if $\sigma_1\sigma_2 Q_1(s) \ge \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_1(s)$, we have:

$$\begin{aligned} &\sigma_1\sigma_2 Q_1(s) - \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_1(s) \\ \le &\sigma_1\sigma_2 Q_1(s) - \sigma_1\sigma_2\hat{Q}_1(s) \\ = &\sum \sigma_1(a_1)\sigma_2(a_2)(Q_1(s,a_1,a_2) - \hat{Q}_1(s,a_1,a_2)) \\ \le &\max_s \sum \sigma_1(a_1)\sigma_2(a_2)(Q_1(s,a_1,a_2) - \hat{Q}_1(s,a_1,a_2)) \\ = &\sum \sigma_1(a_1)\sigma_2(a_2)||Q_1(s,a_1,a_2) - \hat{Q}_1(s,a_1,a_2))|| \\ = &||Q_1(s) - \hat{Q}_1(s))|| \end{aligned} \tag{5.6}$$

for $\sigma_1\sigma_2 Q_1(s) \le \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_1(s)$ the proof is similar to the above, thus

$$||P^t Q_1 - P^t \hat{Q}_1|| \le \gamma ||Q_1(s) - \hat{Q}_1(s))|| \tag{5.7}$$

which means Q_1 will converge to a fixed point. Once Q_1 is converged, σ_1 will also be fixed for all states s, i.e., $\pi_1(s) = \hat{\pi}_1(s)$ ($\sigma_1 = \hat{\sigma}_1$) for all $Q, \hat{Q} \in \mathbf{Q}$

Follower agent(Agent 2)

if $\sigma_1\sigma_2 Q_2(s) \geq \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_2(s)$

$$\begin{aligned}
&\sigma_1\sigma_2 Q_2(s) - \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_2(s)\\
=&\sigma_1\sigma_2 Q_2(s) - \sigma_1\hat{\sigma}_2\hat{Q}_2(s)\\
\leq&\sigma_1\sigma_2 Q_2(s) - \sigma_1\sigma_2\hat{Q}_2(s)\\
=&\sum \sigma_1(a_1)\sigma_2(a_2)(Q_2(s,a_1,a_2) - \hat{Q}_2(s,a_1,a_2))\\
\leq&\max_s \sum \sigma_1(a_1)\sigma_2(a_2)||Q_2(s,a_1,a_2) - \hat{Q}_2(s,a_1,a_2))||\\
=&\sum \sigma_1(a_1)\sigma_2(a_2)||Q_2(s,a_1,a_2) - \hat{Q}_2(s,a_1,a_2))||\\
=&||Q_2(s) - \hat{Q}_2(s))||
\end{aligned} \tag{5.8}$$

for $\sigma_1\sigma_2 Q_2(s) \leq \hat{\sigma}_1\hat{\sigma}_2\hat{Q}_2(s)$ the proof is similar to the above, thus

$$||P^t Q_2 - P^t\hat{Q}_2|| \leq \gamma||Q_2(s) - \hat{Q}_2(s))|| \tag{5.9}$$

Theorem 1 *Under assumptions 1 - 3, the sequence $Q^t = (Q_1^t, Q_2^t)$, updated by*

$$\begin{aligned}
&a_1' = \arg\max_{a_1} Q_1(s', a_1, a_2')\\
&\quad s.t. \quad G_1(s', a_1, a_2') \leq d_1\\
&\qquad a_2' = \arg\max_{a_2} Q_2(s', a_1, a_2)\\
&\qquad\quad s.t. \quad G_2(s', a_1, a_2) \leq d_2\\
&\phi_1 \leftarrow \phi_1 - \alpha_1 \bigtriangledown_{\phi_1} (Q_1(s,a_1,a_2) - r_1 - \gamma(1-d)Q_1^{targ}(s', a_1', a_2'))\\
&\phi_2 \leftarrow \phi_2 - \alpha_2 \bigtriangledown_{\phi_2} (Q_2(s,a_1,a_2) - r_2 - \gamma(1-d)Q_2^{targ}(s', a_1', a_2'))\\
&\zeta_1 \leftarrow \zeta_1 - \beta_1 \bigtriangledown_{\zeta_1} (G_1(s,a_1,a_2) - c_1 - \gamma(1-d)G_1^{targ}(s', a_1', a_2'))\\
&\zeta_2 \leftarrow \zeta_2 - \beta_2 \bigtriangledown_{\zeta_2} (G_2(s,a_1,a_2) - c_2 - \gamma(1-d)G_2^{targ}(s', a_1', a_2'))
\end{aligned} \tag{5.10}$$

in section 4.2 leads to the convergence of the Q-values to a fixed value denoted as $Q^* = (Q_1^*, Q_2^*)$. This convergence is established through two key observations.

First, by applying Lemma 3, it is shown that the update operator P^t is a contraction operator. This property ensures that the Q-values progressively approach a fixed point as the iterations proceed.

Second, the fixed point condition, represented by the equation $E[P^tQ^*] = Q^*$, is established using Lemma 2. This condition indicates that the expected value of applying the update operator to the fixed Q-value results in the same Q-value.

Combining these observations with the insights from Lemma 1, we conclude that the Q-values will converge to Q^* with probability 1. In other words, as the algorithm continues to iterate, the

Q-values approach the optimal solution, ensuring the convergence of the reinforcement learning process.

5.2 Summary

The previous proof of convergence only applies to the constrained Stackelberg Q-learning algorithm, and it is a sufficient but not necessary proof. The conditions assumed in the proof may not always be fully met, making it difficult to provide precise proof of the algorithm's convergence. Furthermore, proving the convergence of the constrained Stackelberg MADDPG algorithm is even more challenging because it involves multiple neural networks, including the actor, critic, cost critic, and Lagrange multipliers. Although we cannot provide a mathematical proof of convergence in this paper, it does not mean that our algorithms cannot converge. We have conducted numerous experiments and have consistently observed that our algorithms can converge and maintain stability after convergence when provided with adequate training data. We will present the experimental results in the following section.

6 Experiment

This section presents the experimental evaluation of the Constrained Stackelberg Q-learning algorithm and the Constrained Stackelberg MADDPG algorithm in a simulation environment called highway-env. These experiments aim to assess the performance and convergence of the algorithms in various autonomous driving scenarios, including merge, intersection, roundabout, and racetrack. In the following sections, we will first introduce the simulation environment library and its basic settings and then present various testing environments and experimental results.

6.1 Environment:Highway-env

Highway-env [28] is a GitHub project that provides a collection of simulation environments designed for autonomous driving, such as highway scenarios. It is developed based on Open AI's Gym environment toolkit [29]. The vehicles are modeled in this environment as the Kinematic Bicycle Model [30]. This classic model can accurately capture vehicle motion in normal driving conditions. And it only requires inputting two variables, front wheel steering angle, and rear wheel acceleration, to control the vehicle's motion easily. This library also provides input-output interfaces to accommodate different reinforcement learning algorithms.

It provides two input methods for the action input: continuous action and discrete action. The continuous action type allows the agent to directly set the low-level controls of the vehicle kinematics, namely the throttle α and steering angle δ. The discrete action control is achieved through an upper-level controller on top of the continuous action control. In the default setting, the user can freely choose an action between 'LANE_LEFT', 'IDLE', 'LANE_RIGHT', 'FASTER', and 'SLOWER', where 'IDLE' means stay unchanged. Once the actions are set, the higher-level controller will automatically control the speed and steering angle to achieve the desired state of the vehicle. For both action types, the longitudinal (speed changes) and lateral (lane changes) actions can be disabled separately through parameter setting.

The reward function design for achieving optimal driving behavior in various scenarios is instead a challenging problem. Therefore, this library only provides a simple implementation of a reward function that is composed of a velocity term and a collision term. However, this library offers many convenient tools and interfaces that allow users to define their reward functions easily. In the definition of the vehicle class, users have free access to various vehicle states, including speed, coordinates, collision status, on-road status, and more. Therefore, users can guide the agent to learn desired behaviors by designing reward functions.

Several types of observations are available for all environments, including Kinematics, Grayscale Image, Occupancy Grid, and Time to Collision. Each environment has a default observation, which can be changed or customized using environment configurations. The most commonly used observation type in this thesis is the kinematic observation. It provides a list of kinematic features such as coordinates (x, y) and velocity (v_x, v_y) of nearby or interested vehicles. The kinematic features can also be configured as absolute or normalized using environment configurations.

However, not all environments are equipped with a multi-agent version and can output the cost signal, which is required to test our algorithms. For this reason, these environments are modified in several ways to accommodate our algorithms and experiments.

6.2 Evaluation metrics

In the context of a safe multi-agent reinforcement learning algorithm designed for autonomous driving scenarios with two vehicles, it is crucial to establish appropriate evaluation metrics that capture the algorithm's safety performance and overall effectiveness. Here, we present a set of evaluation metrics specifically designed to assess the safety and performance of our algorithms.

- **Total accumulated rewards:** For reinforcement learning algorithms, the total accumulated rewards is a universal evaluation metric. This metric quantifies the overall performance of the MARL algorithm throughout the training or evaluation process. It measures the sum of rewards the agents receive over an entire episode. In the context of autonomous driving, rewards can be designed to encourage safe driving behaviors such as staying within speed limits, maintaining proper distance, and obeying traffic signals. On the other hand, it can also penalize unsafe actions that may result in collisions or traffic violations. The accumulated reward reflects the effectiveness of the MARL algorithm in optimizing its behavior to maximize the desired objectives.

- **Success rate:** The success rate metric measures the proportion of episodes in which the agents successfully achieve the desired goal without any safety violations or collisions under the guidance of the algorithms. For our autonomous driving scenarios, a successful episode is defined as all agents completing a designated task (e.g., reaching a destination) while adhering to traffic rules and avoiding accidents. The success rate indicates the algorithm's ability to ensure the safety of agents and environments.

Success Rate $= \frac{\text{Number of Successful Episodes}}{\text{Total Number of Episodes Counted}}$

- **First arrive rate:** First arrive rate metric measures the proportion of episodes in which the leader agent arrives at the destination first or the follower agent arrives first. Since most of our scenarios are designed to be a mixed cooperative-competitive game. The agents must first compete to arrive at the destination to gain a bonus reward. In addition, due to the Stackelberg model, agents' decision-making process is asymmetric. In the Stackelberg model, the leader has been proven to have a comparative advantage over the follower. This metric can help to demonstrate the feature more prominently and validate the correctness of the model application.

- **Convergence:** Convergence is a critical metric that evaluates how well the MARL algorithm converges towards an optimal or near-optimal policy during training. It measures the rate at which the algorithm's performance stabilizes and reaches a consistent level of performance. Convergence can be assessed by monitoring the learning curve of performance metrics, such as accumulated reward or success rate, across multiple training iterations or episodes. A high convergence rate indicates that the algorithm learns and adapts effectively to the autonomous driving environment.

6.3 Baseline Algorithms

Our experiment is mainly divided into two parts. One part is the discrete action scenario(Merge and Roundabout environments), and the other part is the continuous action scenario(Intersection and Racetrack environments). In the discrete action experiment, we compare our algorithm constrained Stackelberg Q-learning with the baseline algorithm Bi-AC. In the continuous action experiment, our algorithm constrained Stackelberg MADDPG needs to be compared with three other algorithms, namely Bi-AC, MACPO, and MAPPO_L [31].

6.4 Merge

6.4.1 Environment settings

This environment consists of two straight main roads and one side road that is about to merge into the main road. The leader agent vehicle starts on one of the main roads next to the side road and will soon approach the road junction. The follower agent vehicle begins on the side road and must merge into the main roads before colliding with the obstacle at the end of the side road. In addition, a finish line is set after the road junction. The initial positions and velocities of the two vehicles are pre-set and added with random noise. A collision is likely to occur if both cars maintain their initial driving states without making significant adjustments. The leader agent's objective in this task is to maintain a high speed while making room for the follower agent so that they can safely merge into the traffic. The follower agent's objective is successfully merging into the main road without colliding with the leader agent or the road obstacle. Each agent will be rewarded if its speed remains within a high range of $[8, 12]$. The first agent to cross the finish line will receive additional rewards than the other agent. If a collision happens, the crashed vehicle will be punished with a high cost signal. Once a car has crashed or crossed the finish line, its state will be marked as terminated. The simulation will end when both vehicles' states are terminated or the simulation time reaches the upper limit. Therefore, this task is a general-sum and mixed cooperative-competitive game. The two agents need to cooperate to reach the finish line safely, but at the same time, they also need to compete to arrive at the finish line first to gain bonus rewards.

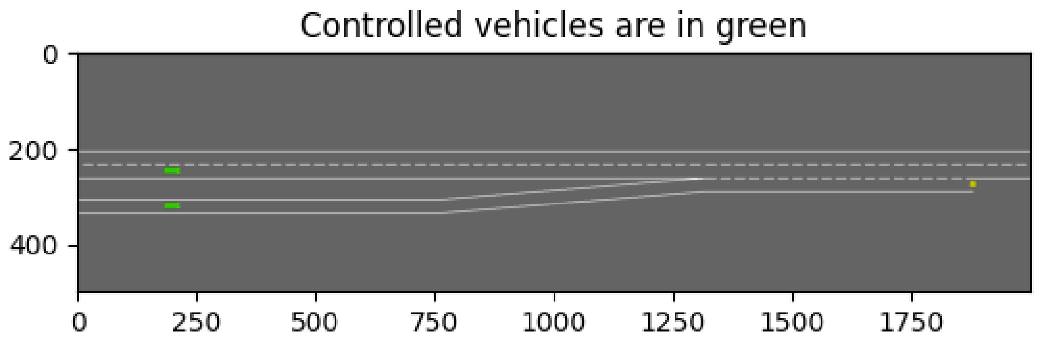

Figure 6.1: An illustration of the Merge Environment

6.4.2 Results

Figures showcasing the performance of Bi-AC and C_SQ in the merge scenario are presented below. These include leader and follower agents' reward curves, as well as the total reward curve. Additionally, training curves displaying the occurrence of three different scenarios during the training process of both algorithms are shown. In Fig 6.2, the rewards obtained by the leader agents of both algorithms converge to approximately 30. Although Bi-AC converges slightly higher than C_SQ, both algorithms exhibit frequent and significant fluctuations in their leader agent reward curves before convergence. In Fig 6.3, the follower agents' rewards converge to around 25 under the guidance of both algorithms. C_SQ obtains higher rewards than Bi-AC, and the follower agent reward curves show a clear and monotonic trend during training. From the total reward curve in Fig 6.4, it is evident that C_SQ outperforms Bi-AC with a certain advantage.

Fig 6.5 shows that in the early stages of training, the chances of collision are significant for both algorithms, reaching up to 60%. This results in significant fluctuations in the reward curves. The likelihood of collisions decreases and converges as training progresses, although Bi-AC still has approximately a 10% chance of collision after convergence. In contrast, C_SQ is not only able to increase the safe rate rapidly but also significantly reduces the probability of collisions after convergence, ensuring complete safety.

There is hardly any evidence of followers leading in Fig 6.5, which may be due to the scenario's design being more favorable for leaders. The leader agent usually only needs to maintain a straight trajectory or perform a single lane change to reach the endpoint successfully. In contrast, the follower agent needs to avoid obstacles and prevent collisions with the leader agent, requiring more complex maneuvers to reach the endpoint safely. Additionally, in the Stackelberg model, the leader's decision-making priority is higher than that of the follower, giving the leader an advantage at the decision-making level. These two factors result in a significantly higher probability of the leader agent reaching the endpoint first than the follower. The convergence reward for followers is lower than the convergence reward for leaders, and the five-point difference represents the bonus reward for reaching the destination first.

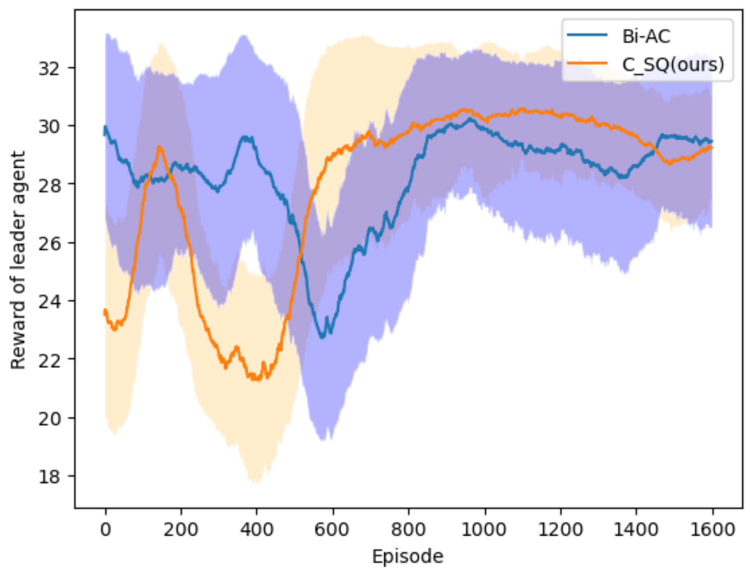

Figure 6.2: Leader agents' reward curves of merge env

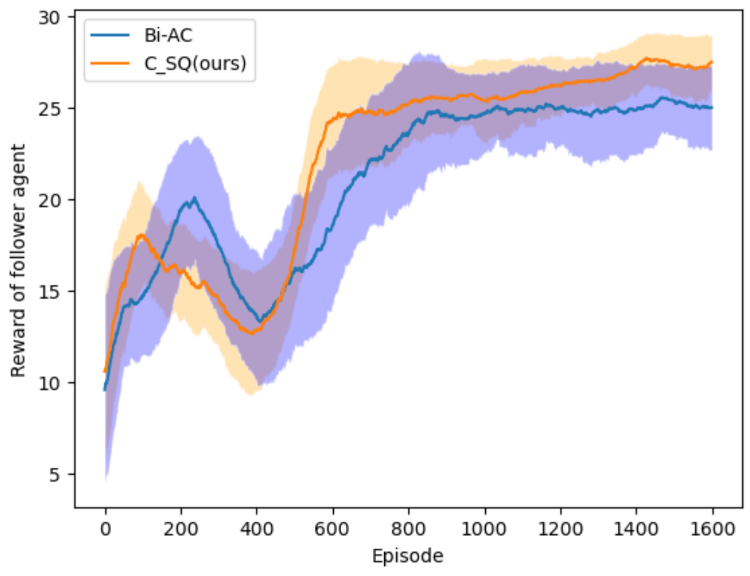

Figure 6.3: Follower agents' reward curves of merge env

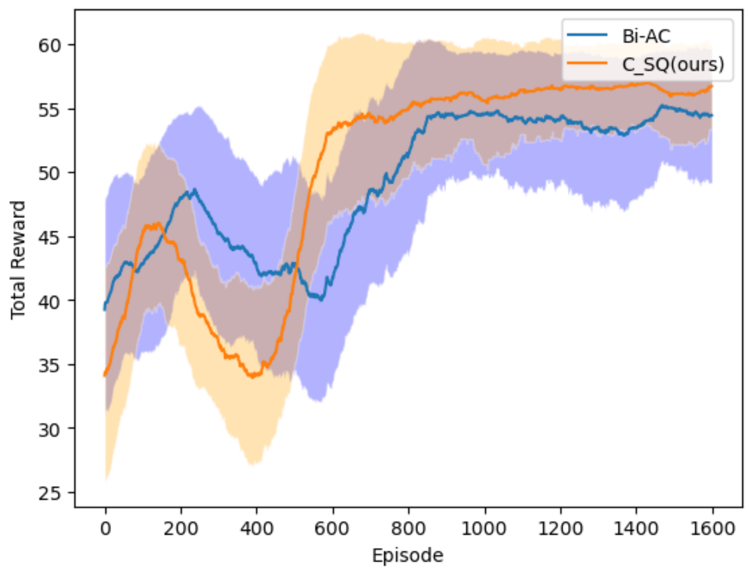

Figure 6.4: Total reward curves of merge env

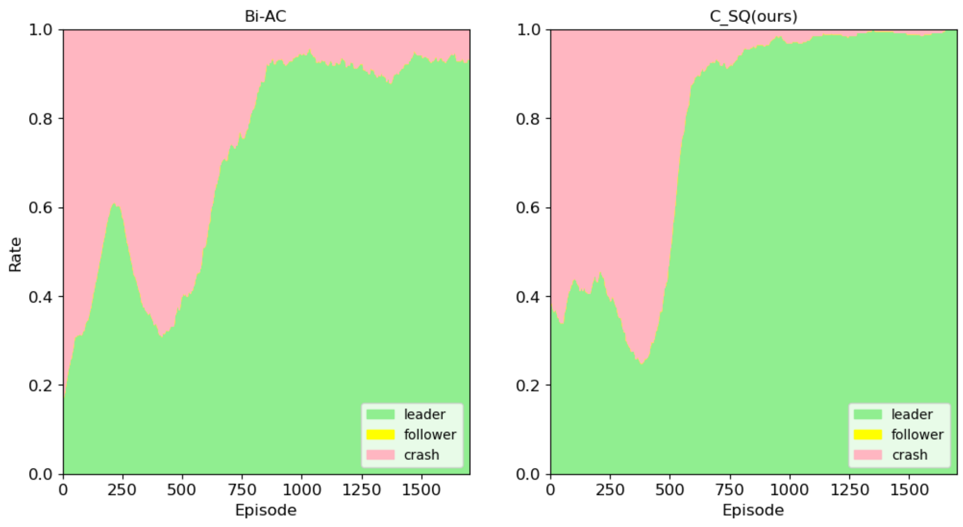

Figure 6.5: The training curves for the merge environment are shown in the left and right figures, corresponding to the Bi-AC and C_SMADDPG algorithms, respectively.

6.5 Roundabout

6.5.1 Environment settings

This scenario is a circular intersection or junction with a roundabout consisting of two lines 6.6. The roundabout has four entrances/exits located in the north, south, east, and west directions of the ring, with the leader agent vehicle spawning before the west entrance and the follower agent vehicle spawning before the south entrance. If both vehicles maintain their current speed and lane, they will meet at the south entrance, which could result in a collision. Both vehicles will automatically follow their planned

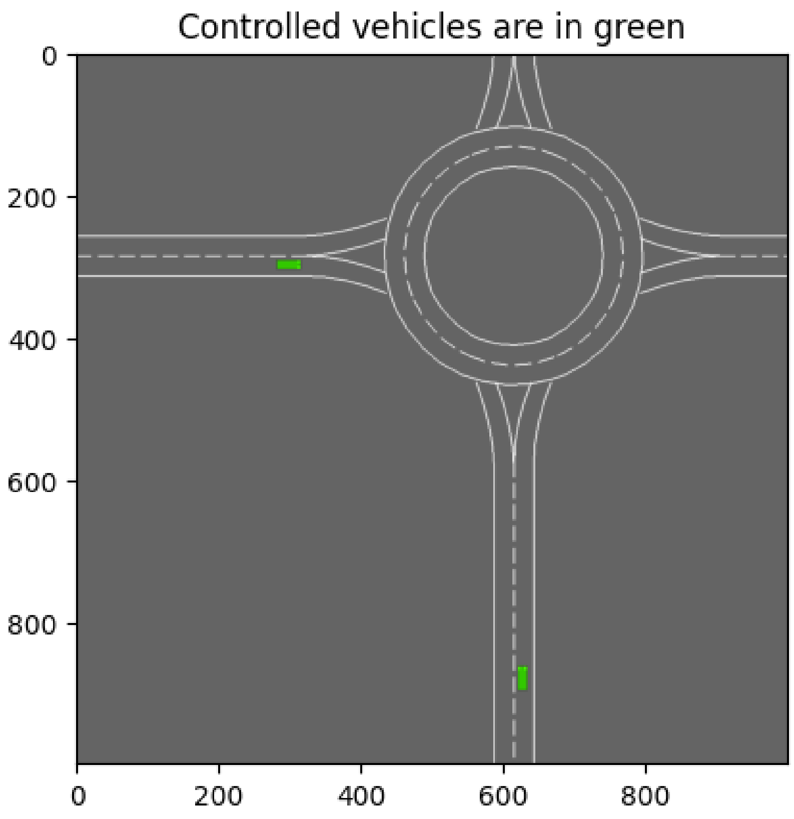

Figure 6.6: An illustration of the Roundabout Environment

route, which leads to their destination shortly after the north exit. However, they must also handle lane changes and longitudinal control to pass through the roundabout as quickly as possible while avoiding collisions. The agents will be rewarded if they follow the traffic rule by maintaining their velocity within a set range of $[8, 12]$. The agent that reaches the finish line first will receive a higher reward than the other vehicle.

There are no obstacles in this scenario, and a collision will only occur if the two vehicles collide. Then both vehicles will be punished with high cost signals. If a collision occurs or a car crosses the finish line, its state will be marked as terminated. The simulation will end when both vehicles' states are terminated, or the simulation time reaches the upper limit. This task is a general-sum and mixed cooperative-competitive game, where the two cars must cooperate and allocate the road to navigate the roundabout safely while also seizing opportunities to prioritize exiting and gain higher rewards.

6.5.2 Results

Figures displaying the reward curves and training curves for algorithms Bi-AC and C_SQ in the roundabout experiment are presented below. The reward curves for both the leader and follower agents in Fig 6.7 and Fig 6.8 clearly show a steady increase with evident monotonic trends. Once convergence is reached, the rewards stabilize, with the leader agents' reward plateauing at around 33. C_SQ only exhibits a marginal advantage of 3 points over Bi-AC. The follower agents' reward curves for both algorithms in Fig 6.8 are almost identical, converging at 30. Hence, in terms of the total reward, C_SQ is slightly ahead of Bi-AC.

In the final training curve plot shown in Fig 6.10, the safety probabilities of both algorithms increase steadily and converge during the later stages of training. However, as in the previous experiment, Bi-AC still has a collision rate of approximately 5% after convergence, whereas C_SQ ensures 100% safety for both vehicles once convergence is reached. This is also reflected in the reward curves, where C_SQ has a slightly higher overall sum of rewards for both agents. This can be attributed to the cost constraints enforced by C_SQ, which exclude action choices that could result in collisions.

Moreover, it is observed that in the early stages of training, there are instances where the follower agent reaches the destination first. However, this proportion gradually decreases during training, with the leader agent becoming dominant in reaching the destination first. In the current experiment setting, both the leader and follower agents are in similar conditions, with no clear advantage for either side to reach the destination earlier. However, in the Stackelberg model, the leader agent exploits the decision-making priority to maximize its rewards.

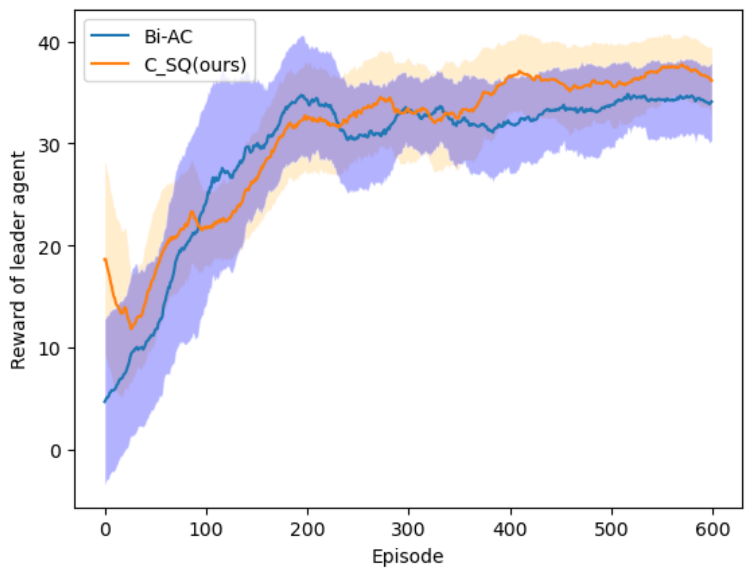

Figure 6.7: Leader agents' reward curves of roundabout env

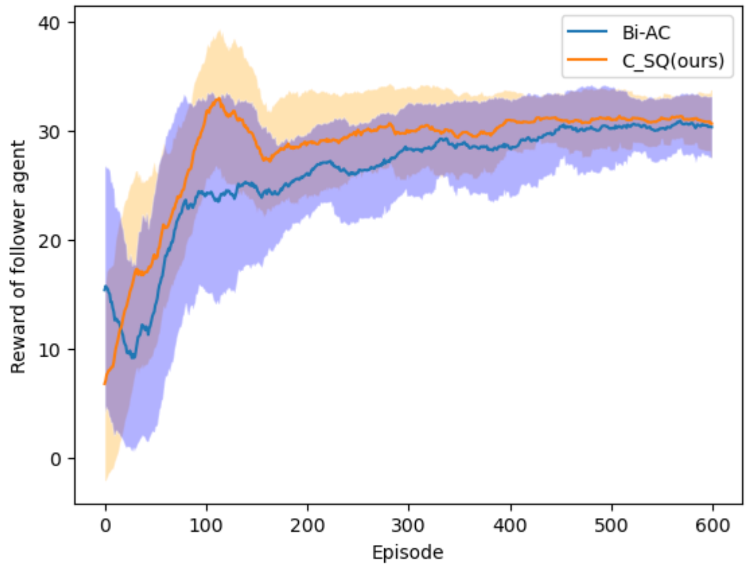

Figure 6.8: Follower agents' reward curves of roundabout env

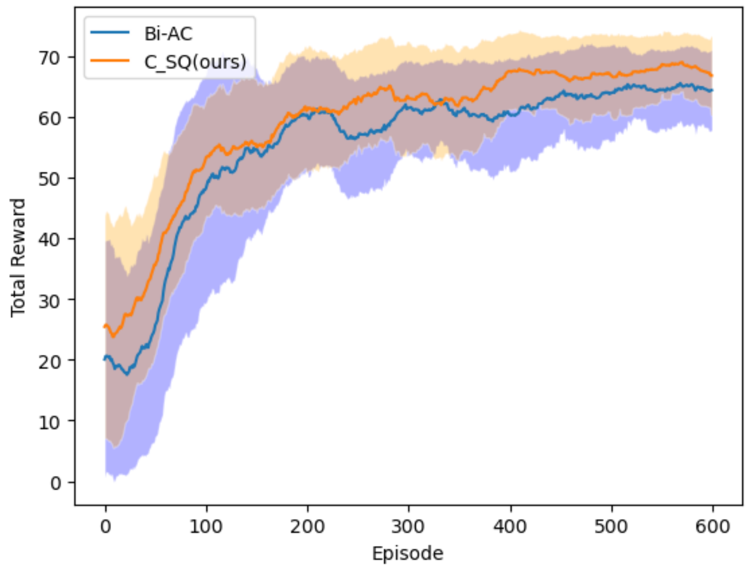

Figure 6.9: Total reward curves of roundabout env

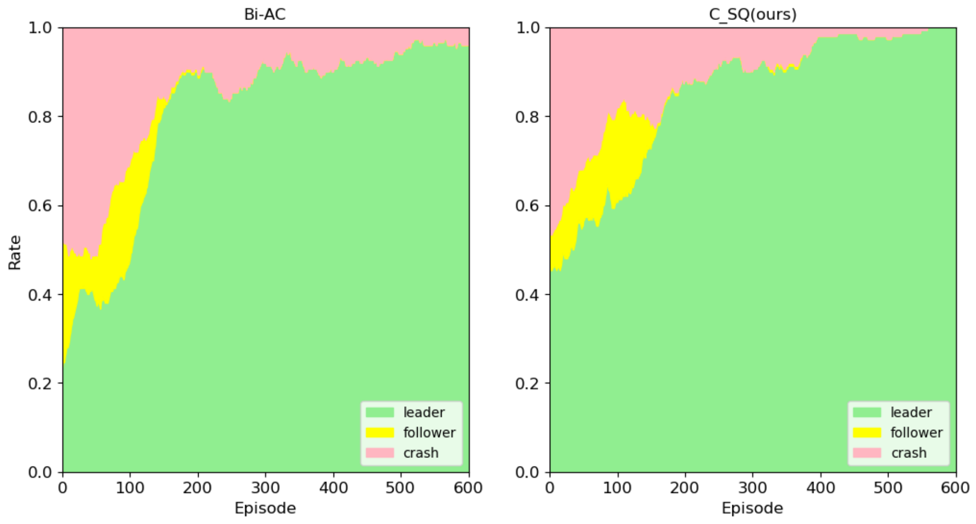

Figure 6.10: The training curves for the roundabut environment are shown in the left and right figures, corresponding to the Bi-AC and C_SMADDPG algorithms, respectively.

6.6 Intersection

6.6.1 Environment settings

This scenario is an intersection where two roads intersect at right angles, forming a four-way junction. Vehicles approach the intersection from each of the four directions: north, south, east, and west. Each vehicle aims to pass through the intersection and reach its destination on the other side. However, this intersection has no traffic lights to control the traffic. Due to the absence of traffic lights, the vehicles have to navigate themselves to pass this intersection safely. In this test, we only consider two agents. The leader agent vehicle intends to travel from the south to the north, while the follower agent vehicle intends to travel from the west to the east. Therefore, they will soon meet at the intersection. For this situation, a possible solution is that one of the vehicles slows down before the intersection while the other accelerates to pass through. Based on their observations, The two agents must independently decide which action to take and figure out a proper throttle variable to control the vehicles. Except for collision avoidance, the two agents must also ensure that their driving velocities are within a safe range [8,12]. Setting a velocity upper limit prevents vehicles from overspeeding and causing traffic accidents, while the lower limit reduces traffic congestion. The agents will get rewarded if they follow this traffic rule or get nothing. To add competition, a bonus reward is set to guide the agents to arrive at their destination first. However, they must not crash into each other, or they will both be punished with high cost signals.

6.6.2 Result

In our scenario that involves continuous action space control, we have introduced new baselines, namely MACPO and MAPPO_L, for comparative experiments. The reward and training curves for the four algorithms are displayed in the following Figures Fig 6.12, Fig 6.13, Fig 6.14, and Fig 6.15. Fig 6.12 shows that the leader agent reward curves for each algorithm are quite scattered. Our C_SMADDPG algorithm ranks second, with a convergence point of around 16, which is higher than Bi-AC but lower than MACPO, albeit by a small margin of just two points. Looking at the follower agent reward curves in Fig 6.13, the MAPPO_L algorithm has the highest convergence point at 16 points, while our C_SMADDPG algorithm still ranks second at 15. Furthermore, regarding the total reward sum after convergence for both leader and follower agents, the reward curves for C_SMADDPG and MACPO are nearly identical, reaching 32, ranking first. Bi-AC and MAPPO_L achieve lower rewards, at 30 and 26, respectively, as shown in Figure 6.14.

Fig 6.15 presents the event distribution during the training process for each algorithm. It is observed that collisions consistently occur throughout the entire training process

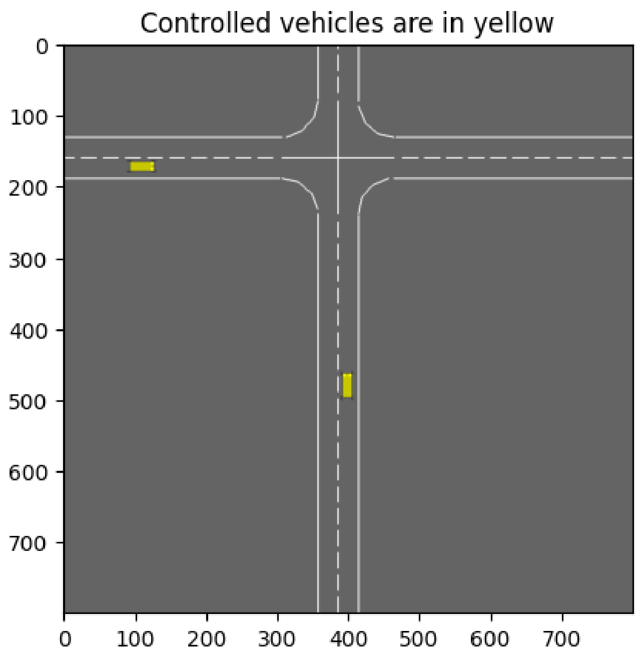

Figure 6.11: An illustration of the Intersection Environment

in the Bi-AC algorithm, as there are no cost constraints. Even after convergence, the collision probability remains as high as 10%. However, the other three algorithms benefit from the cost constraints, resulting in a noticeable reduction in collision probability during the training process. The MAPPO_L algorithm achieves the overall highest safety rate, reaching a 95% safety rate early in training and quickly improving to 100% until the end of training. Upon reviewing the simulation videos, it was observed that in the MAPPO_L training process, the two agents typically have a significant speed difference, with one vehicle rapidly passing through the intersection at a higher speed while the other vehicle moves slowly. Such a significant speed difference dramatically reduces the chances of collision and prevents the agents from obtaining speed rewards, resulting in the lowest reward for the MAPPO_L algorithm.

The training process for the MACPO algorithm and our algorithm C_SMADDPG is relatively similar. Both cars try to control their speed within the reward range, accelerating and decelerating, respectively, to avoid collisions when passing through the intersection. The safety rate gradually improves during the training process. However, our algorithm can achieve a 100% safety rate.

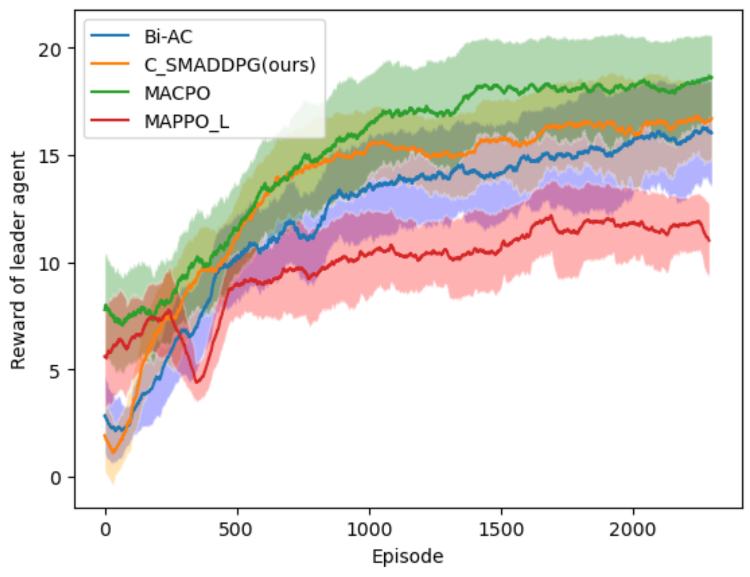

Figure 6.12: Leader agents' reward curves of roundabout env

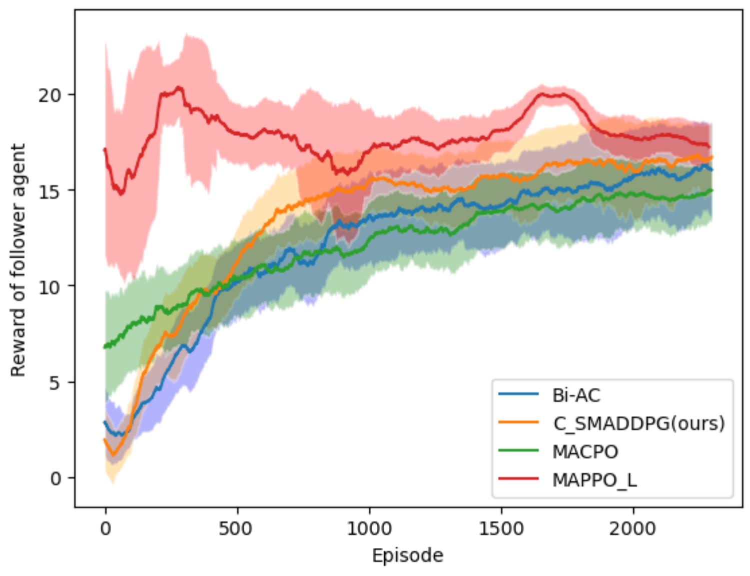

Figure 6.13: Follower agents' reward curves of roundabout env

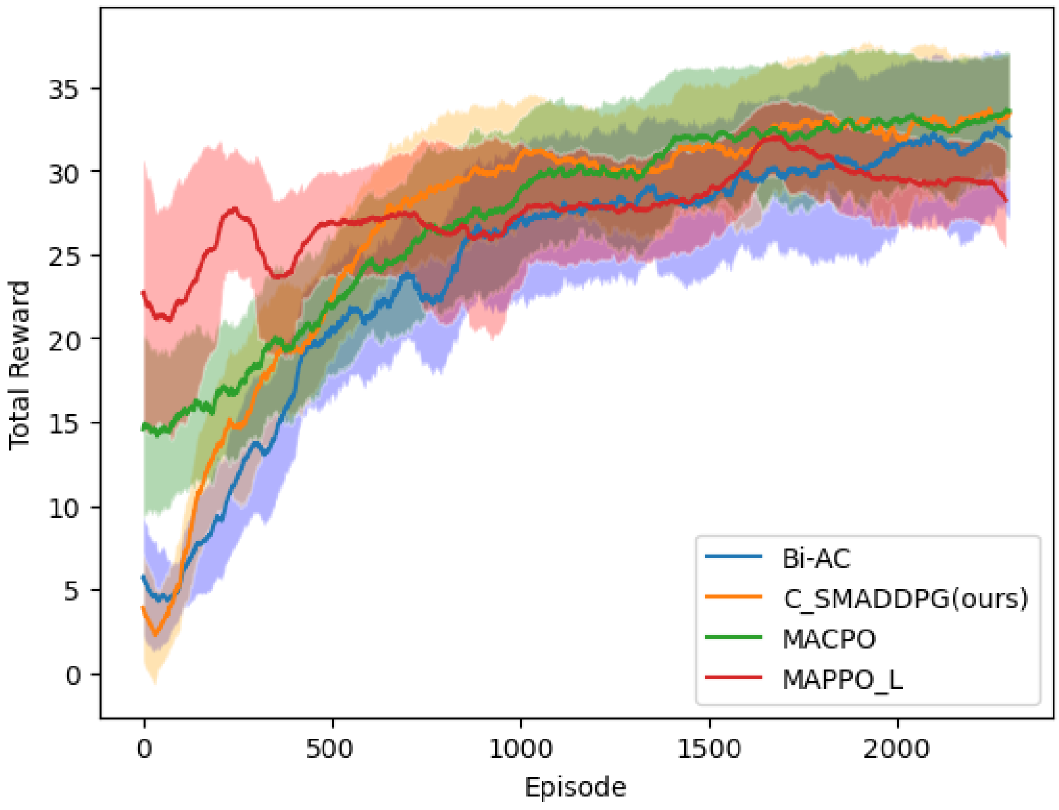

Figure 6.14: Total reward curves of roundabout env

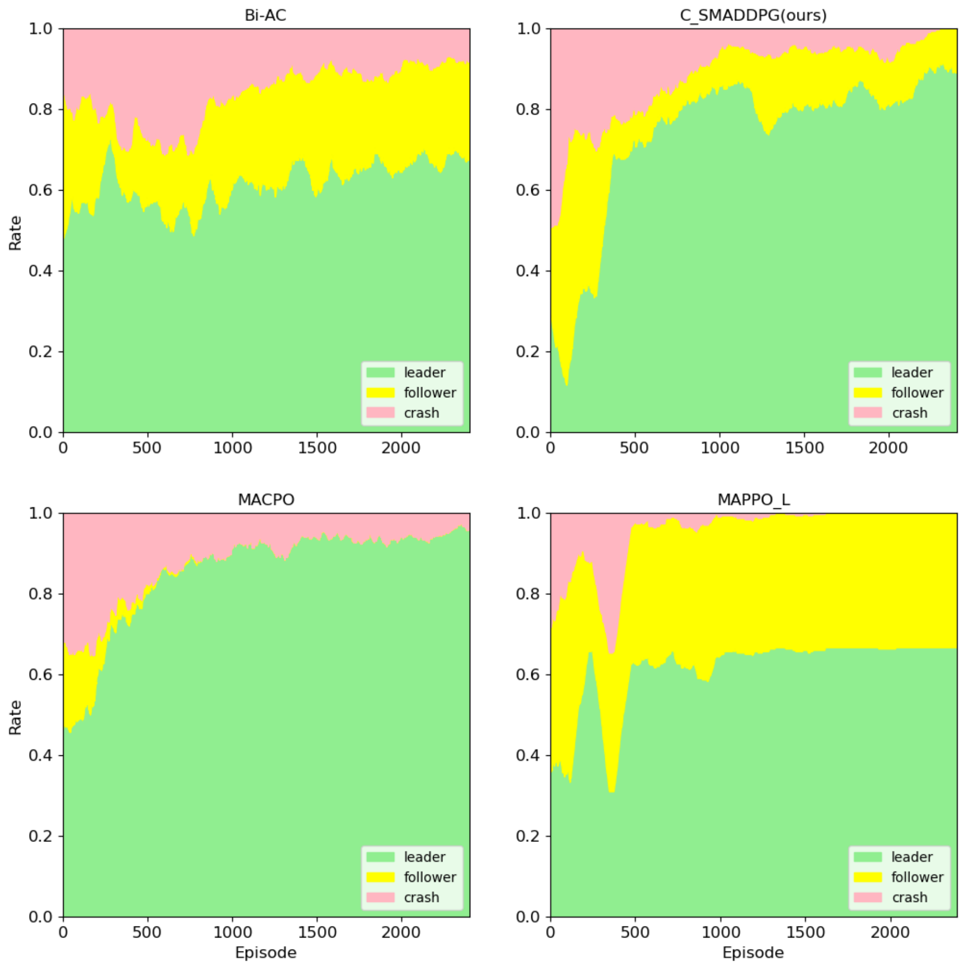

Figure 6.15: The training curves for the intersection environment are displayed in the figures below. The top left and top right figures represent the training curves for Bi-AC and C_SMADDPG algorithms, respectively. The bottom left and bottom right figures depict the training curves for MACPO and MAPPO_L algorithms, respectively.

6.7 Racetrack

6.7.1 Environment settings

The racetrack environment consists of a circular track with two lanes, shown in Fig 6.16, featuring straight sections and curves that make it more complex than previous environments. This is a continuous control environment where agents only need to control their steering angles and maintain a constant longitudinal velocity. The leader and follower agent vehicles begin parallel to each other at the starting point on the straight track, and their goal is to navigate the track while avoiding collisions with other vehicles and staying within the lane boundaries. Collisions are highly probable if the agents fail to control their steering angles properly, as they will always be very close to each other due to the constant longitudinal velocity. The vehicle earns the highest reward for staying in the center lane, and the rewards decrease as it deviates from the center line. If the vehicle leaves the lane, it receives no rewards. Collisions result in high-cost penalties. The agents do not have a destination or finish line to reach but must safely travel along the lane throughout the simulation time. If a vehicle crashes or leaves the road, it is marked as terminated, and the simulation ends when both agents are in a terminated state or when the simulation time reaches its limit. This is a cooperative game where the agents must learn to follow the track and avoid collisions to achieve maximum rewards within the given time.

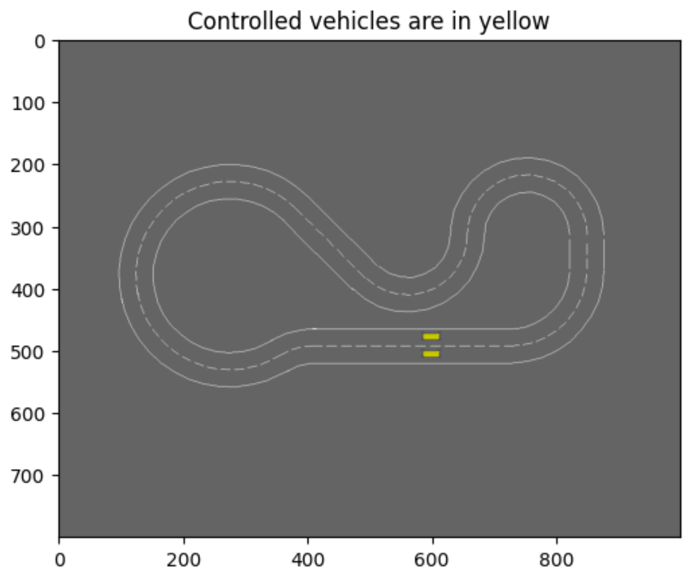

Figure 6.16: An illustration of the Racetrack Environment

6.7.2 Result

The racetrack environment remains using continuous action space. We compared our C_SMADDPG algorithm with Bi-AC, MACPO, and MAPPO_L as baselines. The reward curves and training curves are shown in Fig 6.17 Fig 6.18, Fig6.19 ,and Fig 6.20. The trends shown in the three reward curves are almost identical. Our C_SMADDPG algorithm achieves the highest rewards and the fastest rate of increase. The total rewards of C_SMADDPG can reach 150 after training, which is about three times higher than the other three baselines. However, similar to MAPPO_L, there is significant fluctuation during training. Both of these algorithms use Lagrange multipliers, and this instability may be caused by the varying Lagrange multipliers, which result in the optimization objective function constantly changing, leading to training instability. Additionally, while the reward curves of Bi-AC and MACPO algorithms steadily increase, the improvement rate is slow, and they cannot achieve high rewards.

Fig 6.20 shows the training curves of the four algorithms in this experiment. Unlike before, this experiment only recorded safety and collisions as events. Safety indicates that no collisions occurred between the two vehicles until the end of the simulation. Since no endpoint was set in this experiment, both agents only needed to drive safely along the track. From this figure, it can be observed that Bi-AC performed poorly in the absence of cost constraints. Due to the complex terrain in this experiment, the vehicles needed to make continuous turns to adjust their direction, resulting in a high probability of collisions. After convergence, the Bi-AC algorithm had only a 40% chance of avoiding collisions. This led to very low rewards for both the leader and follower agents in the Bi-AC algorithm. MACPO had the highest and most stable safety rate throughout the training, maintaining around 80%. Since the initial stages of the track are straight, MACPO enables the vehicle to quickly learn to drive in parallel lines to avoid collisions. As a result, the safety rate steadily increases. However, the vehicle continues to drive straight and eventually exits the track, causing the simulation to stop, and the obtained rewards are relatively low. Our C_SMADDPG and MAPPO_L algorithms exhibit significant fluctuations in safety rate, with occasional highs and lows, as agents explored the curved environment and frequently experienced collisions upon encountering new environments. However, overall, the C_SMADDPG algorithm demonstrates better safety performance than MAPPO_L, maintaining a safety rate of approximately 80% during training.

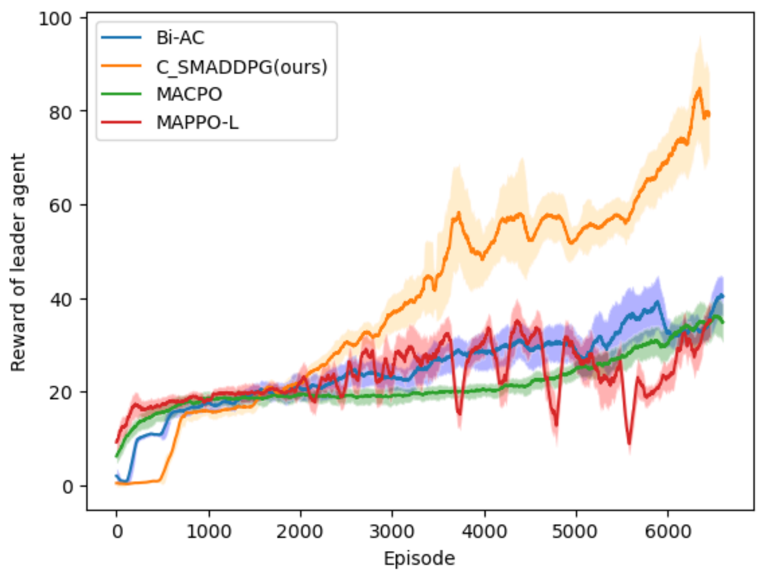

Figure 6.17: Leader agents' reward curves of racetrack env

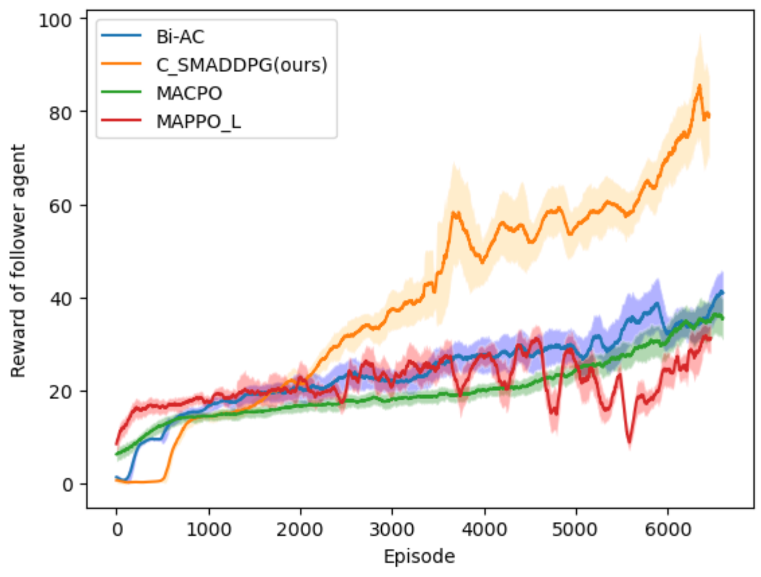

Figure 6.18: Follower agents' reward curves of racetrack env

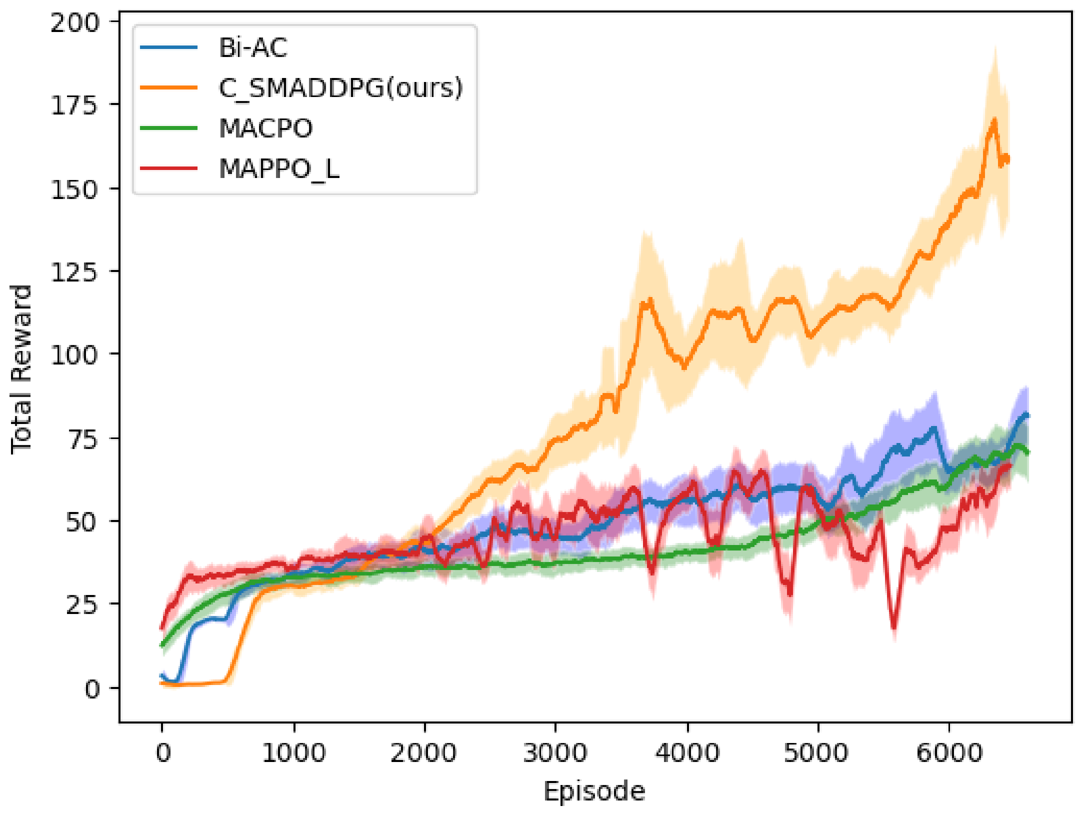

Figure 6.19: Total reward curves of racetrack env

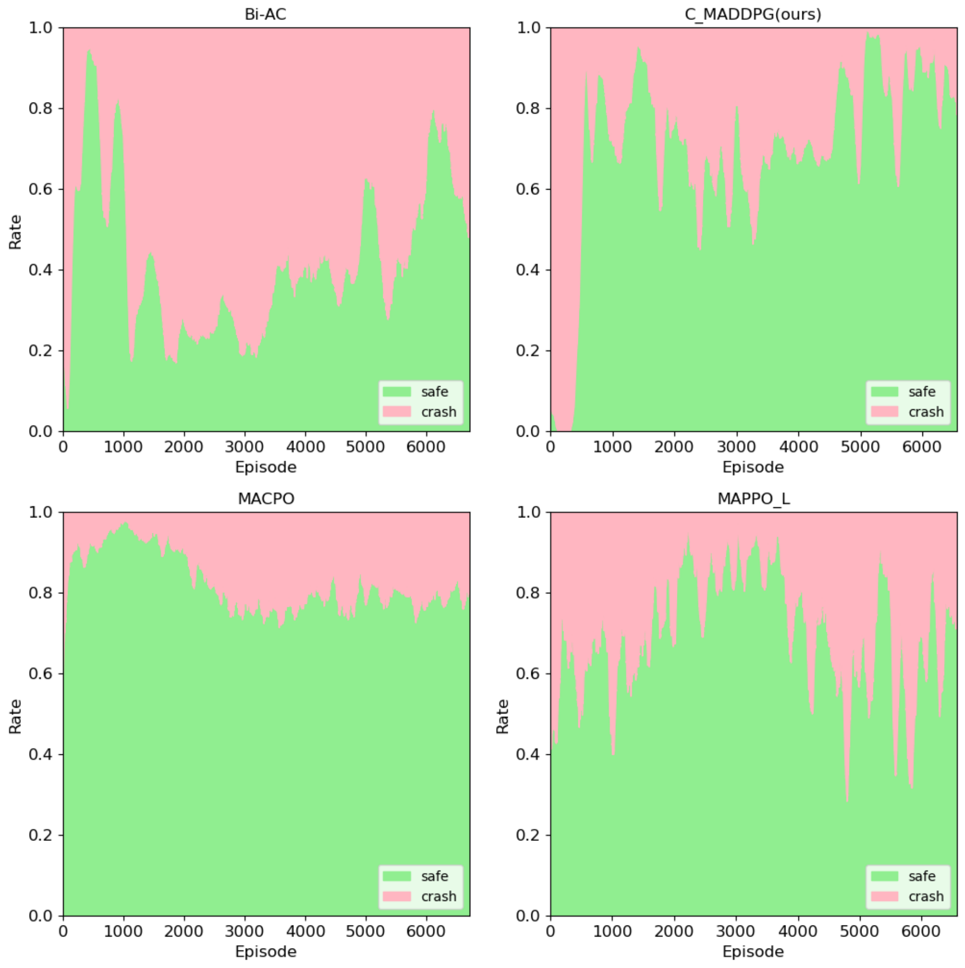

Figure 6.20: The training curves for the racetrack environment are displayed in the figures below. The top left and top right figures represent the training curves for Bi-AC and C_SMADDPG algorithms, respectively. The bottom left and bottom right figures depict the training curves for MACPO and MAPPO_L algorithms, respectively.

7 Conclusions & Future Directions

7.1 Conclusion

In this thesis, we explored integrating safe reinforcement learning techniques into autonomous driving scenarios. By incorporating the Stackelberg model with the Q-learning and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm and leveraging the Lagrangian method to control safety constraints, we developed an approach that promotes safe interactions between multiple autonomous vehicles while optimizing their objectives.

Introducing the constrained Stackelberg model into Q-learning provides a valuable framework for safe reinforcement learning. Applying the Stackelberg model allows for hierarchical decision-making, enabling autonomous agents to consider the strategic behaviors of other agents and make informed choices. This approach has proven effective in ensuring safe interactions between agents in discrete action spaces.

To address the challenges associated with continuous action spaces, we have developed C_SMADDPG. This algorithm integrates the Stackelberg model and leverages the power of deep neural networks to handle continuous action spaces. By utilizing the Lagrangian method, C_SMADDPG effectively incorporates safety constraints into the learning process, striking a balance between performance optimization and maintaining safety.

Through extensive simulations and evaluations of various autonomous driving scenarios, we demonstrated the effectiveness of our proposed algorithms. The results showed that our algorithms outperformed the baseline. Our algorithms effectively addressed the challenges of coordinating multiple autonomous vehicles in complex environments, leading to improved safe rates, total reward, and the ability to adapt to dynamic situations.

7.2 Future directions

However, there are several areas that warrant further exploration and research. One important future direction is the investigation of more advanced optimization techniques and algorithms to improve the efficiency and convergence of the approach. Although the Lagrangian method used in this thesis has proven effective in enforcing safety constraints, there is room for integrating state-of-the-art optimization algorithms.

Additionally, the experiments conducted in this thesis primarily focus on relatively simple autonomous driving scenarios. To fully evaluate the potential capabilities of the proposed algorithms, it is crucial to extend the testing to more complex environments. These could include densely populated urban settings, challenging road conditions, or dynamic traffic scenarios. By subjecting the algorithms to such environments, their adaptability and robustness can be thoroughly examined.

While this thesis focuses on two-agent scenarios, real-world autonomous driving often involves multiple agents interacting simultaneously. Extending the proposed algorithms to handle multi-agent scenarios is a natural progression. Future research can explore techniques to handle coordination, cooperation, and competition among multiple autonomous vehicles. This extension would provide valuable insights into the scalability and effectiveness of the proposed algorithms in complex and realistic driving scenarios.

In conclusion, this thesis contributes to the field of safe reinforcement learning in autonomous driving by proposing an approach that combines the Stackelberg model, MADDPG, and the Lagrangian method. The results demonstrate the algorithm's ability to promote safety while achieving satisfactory performance in various autonomous driving scenarios. The future directions outlined here pave the way for further advancements in safe multi-agent reinforcement learning and its application to real-world autonomous systems.

List of Figures

Bibliography

[1] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, no. null, 1039–1069, 2003, ISSN: 1532-4435.

[2] P. Huang, M. Xu, F. Fang, and D. Zhao, *Robust reinforcement learning as a stackelberg game via adaptively-regularized adversarial training*, 2022. arXiv: 2202.09514 [cs.LG].

[3] H. Zhang, W. Chen, Z. Huang, *et al.*, "Bi-level actor-critic for multi-agent coordination," *CoRR*, vol. abs/1909.03510, 2019. arXiv: 1909.03510. [Online]. Available: http://arxiv.org/abs/1909.03510.

[4] L. Meng, J. Ruan, D. Xing, and B. Xu, "Learning in bi-level markov games," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9892747.

[5] R. Naveiro and D. R. Insua, *Gradient methods for solving stackelberg games*, 2019. arXiv: 1908.06901 [cs.GT].

[6] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015. [Online]. Available: http://jmlr.org/papers/v16/garcia15a.html.

[7] S. Gu, L. Yang, Y. Du, *et al.*, *A review of safe reinforcement learning: Methods, theory and applications*, 2023. arXiv: 2205.10330 [cs.AI].

[8] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, *Risk-constrained reinforcement learning with percentile risk criteria*, 2017. arXiv: 1512.01629 [cs.AI].

[9] T. M. Moldovan and P. Abbeel, *Safe exploration in markov decision processes*, 2012. arXiv: 1205.4810 [cs.LG].

[10] W. Uther, "Markov decision processes," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010.

[11] E. Altman, *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[12] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*, Elsevier, 1994, pp. 157–163.

[13] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[14] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017. arXiv: 1502.05477 [cs.LG].

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, 2014, pp. 387–395. [Online]. Available: https://proceedings.mlr.press/v32/silver14.html.

[19] A. Nowé, P. Vrancx, and Y.-M. De Hauwere, “Game theory and multi-agent reinforcement learning,” in *Reinforcement Learning: State-of-the-Art*, M. Wiering and M. van Otterlo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 441–470.

[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *CoRR*, vol. abs/1706.02275, 2017. arXiv: 1706.02275. [Online]. Available: http://arxiv.org/abs/1706.02275.

[21] S. Gu, J. Grudzien Kuba, Y. Chen, *et al.*, “Safe multi-agent reinforcement learning for multi-robot control,” *Artificial Intelligence*, vol. 319, p. 103 905, 2023, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2023.103905. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370223000516.

[22] C. Tessler, D. J. Mankowitz, and S. Mannor, “Reward constrained policy optimization,” *CoRR*, vol. abs/1805.11074, 2018. arXiv: 1805.11074. [Online]. Available: http://arxiv.org/abs/1805.11074.

[23] D. P. Bertsekas, “Nonlinear programming,” *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[24] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained policy optimization,” *CoRR*, vol. abs/1705.10528, 2017. arXiv: 1705.10528. [Online]. Available: http://arxiv.org/abs/1705.10528.

[25] H. Von Stackelberg, *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[26] B. Colson, P. Marcotte, and G. Savard, "A trust-region method for nonlinear bilevel programming: Algorithm and computational experience," *Computational Optimization and Applications*, vol. 30, pp. 211–227, Mar. 2005. DOI: 10.1007/s10589-005-4612-4.

[27] J. F. Bard and J. E. Falk, "An explicit solution to the multi-level programming problem," *Computers Operations Research*, vol. 9, no. 1, pp. 77–100, 1982, ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(82)90007-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0305054882900077.

[28] E. Leurent, *An environment for autonomous driving decision-making*, https://github.com/eleurent/highway-env, 2018.

[29] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[30] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818. DOI: 10.1109/IVS.2017.7995816.

[31] S. Gu, J. G. Kuba, M. Wen, *et al.*, "Multi-agent constrained policy optimisation," *CoRR*, vol. abs/2110.02793, 2021. arXiv: 2110.02793. [Online]. Available: https://arxiv.org/abs/2110.02793.