Safe Reinforcement Learning for Reliable Systems

March　２０２１

Akifumi Wachi

Safe Reinforcement Learning for Reliable Systems

Degree Programs in Systems and Information Engineering

University of Tsukuba

March　２０２１

Akifumi Wachi

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Akifumi Wachi
February 2021

Acknowledgements

I'm grateful to my advisor Professor, Jun Sakuma, for the support throughout my PhD study. He gave me the vision on how to view my research work from a long-term, bird's-eye view. Under the current COVID-19 situations, his flexible way of communications are very comfortable for me.

I would like to sincerely thank Prof. Hiroyuki Kitagawa, Prof. Kazuhiro Fukui, Prof. Yohei Akimoto, Prof. Yukino Baba, and Prof. Takayuki Osa who are on my research committee, for their helpful discussions and suggestions.

A part of this thesis in based on the collaboration with Yanan Sui. I really appreciate his continuous and helpful discussion and support.

My PhD study has been conducted while working at IBM. I would like to thank all the colleagues in IBM Research AI for letting me pursue this PhD degree.

Last, I would like to give my deepest gratitude to my family for their endless love and support.

Abstract

Autonomous agents are playing an ever-increasing role, including in automated driving vehicles or house-care robots. When an autonomous agent interacts with humans, one of the biggest requirement is *safety*. Especially in the autonomous driving examples, even a single mistake might lead to terrible results; hence it is extremely important to guarantee the soundness of the system after deployment at the development stage.

With the advancement of AI or machine learning technologies, the role of software is becoming more and more complex and sophisticated. Automated driving systems need to deal with extremely complex software involving image recognition, decision making, etc. In order to improve the reliability and safety of software, construction and validation must be steadily carried out in the development stage.

On the other hand, reinforcement learning is a promising paradigm for optimizing an agent's policy and has made significant advances in recent years in many domains. Unfortunately, however, conventional reinforcement learning studies have been ignored safety issues; hence, it is difficult to apply reinforcement learning techniques in safety-critical systems because an agent is likely to execute unsafe actions for the environment or the agent itself.

Therefore, we proposed a reinforcement learning algorithm that incorporates the agent's safety while seeking to optimize the policy maximizing the cumulative reward (i.e., utility). This algorithm provides theoretical guarantees for both reward and safety; that is, with high probability, the agent is guaranteed to achieve a near-optimal policy while executing only safe actions even during training phase as well as inference phase.

In addition, under the consideration that reinforcement learning would be useful not only for software construction but also software validation, we also propose a method to validate an agent using multi-agent adversarial reinforcement learning. This method is not for training an agent safely but for validate a target (either RL-based or rule-based) agent by training other agents using adversarial reinforcement learning.

In this thesis, we discuss how to construct and validate the policy of an autonomous agent for making it work properly in safety-critical applications, while leveraging reinforcement learning techniques.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

α	scaling factor for reward
β	scaling factor for safety
γ	discount factor
$\mathcal{A}$	action space
$\mathcal{L}$	loss function
$\mathcal{S}$	state space
μ	deterministic policy
ω	transition noise
π	policy
θ	parameter
a	action
$d(\cdot,\cdot)$	distance metric
f	transition
g	safety function
h	safety threshold
J	objective function
K	kernel matrix

k^g	kernel for safety
k^r	kernel for reward
L	Lipschitz constant
Q	Q-function
r	reward function
s	state
S_0	initial safe space
t	time step
V	value function

Acronyms / Abbreviations

AdvRA	Adversarial Reward Allocation
BEB	Bayesian Exploration Bonus
CI	Contributor Identification
CPO	Constrained Policy Optimization
DDPG	Deep Deterministic Policy Gradient
DEM	Digital Elevation Model
DOM	Document Object Model
DPG	Deterministic Policy Gradient
ES^2	Early-stopping of Exploration of Safety
ESwF	Experience Storing with Filtration
GAN	Generative Adversarial Net
GNN	Graph Neural Network
GP	Gaussian Process
KVfD	Key Variables for Diversity

MADDPG	Multi-agent Deep Deterministic Policy Gradient
MARL	Multi-agent Reinforcement Learning
MDP	Markov Decision Process
MPC	Model Predictive Control
NPC	Non-player Character
P-ES^2	Practical Early-stopping of Exploration of Safety
PAC-MDP	Provably Approximately Correct Markov Decision Process
PG	Policy Gradient
POMDP	Partially-observable Markov Decision Process
PSRBP	Prioritized Sampling by Replay Buffer Partition
RBF	Radial Basis Function
Reg-KVfD	Registered Key Variables for Diversity
RKHS	Reproducing Kernel Hilbert Space
RL	Reinforcement Learning
SafeExpOptMDP	Safe Exploration and Optimization of MDP
SNO-MDP	Safe Near-optimal Markov Decision Process

Chapter 1

Introduction

Automobile and aircraft systems are called safety-critical systems [60] because their safety is directly related to a serious accident. Recently, automated driving systems and advanced driving support systems, which are typical examples of safety-critical systems, have been actively researched and developed, and the importance of the software for these systems has been attracting a great deal of attention. The research problem to be addressed in this dissertation is,

How should we deal with safety in safety-critical systems?

Our ultimate objective is to deploy a *safe* system in a real environment. Safety is a broad concept; hence it is associated with various aspects of the system (e.g., hardware, software). In this dissertation, we focus on the policy (i.e., decision-making algorithm) of the agent. In other words, we ignore other aspects of the system. For example, we do not handle hardware or image-recognition sub-system.

The process until the system is deployed in the real environment can be generally classified into mainly two steps; that is, 1) *construction* phase and 2) *validation* phase. To enhance the probability of the agent working properly in real environments, both construction and validation phases should be conducted properly. In other words, it is significant to design a system with a safe policy of the agent in the construction phase, and then test it to find bugs or defects in validation phase.

Reinforcement learning (RL, [79]) has become increasingly popular as a framework for learning optimal control policies directly by interacting with an a priori unknown environment, while recently making significant progress as being represented by robotic manipulation [41, 32, 26, 27] and mastering games [69, 39, 51, 70]. The paradigm of reinforcement learning is simple yet powerful, and it is known that the resulting policy tends to be more efficient and robust. However, conventional RL algorithms were agnostic to safety. Because

RL require agents to explore the environment first, random actions are likely to be executed while learning the environment. This is a critical problems especially when the application target is safety-critical such as autonomous driving and medical applications.

Construction of a safe policy

In the field of optimal control theory, there is a significant body of work on safety-constrained decision-making [22, 65, 71, 14] under the problem setting that the "model" (i.e., state transition and reward) is known a priori. Basically, safety is interpreted as *stability*, and safety constraints can be implemented by carefully designing asymptotically stable control [34], and one of the most notable methods is robust control [93, 25, 19]. For example, Aswani et al. [5] considers all the admissible disturbances and obtains the tunes around the nominal trajectory.

In RL settings where state transition function and/or reward function are *unknown* a priori, Altman [2] considers constraints on expected performance under the name of constrained Markov decision processes (CMDP). CMDPs have been extended to continuous state and action spaces as being represented by trust region policy optimization (TRPO, [64]) or constrained policy optimization (CPO, [1]). For example, Achiam et al. [1] and Chow et al. [15] aimed to achieve a safe policy while constraining the Kullback-Leibler (KL) divergence of the old and new policy. Also, Howard and Matheson [30] and Marcus et al. [47] introduced risk-sensitivity in risk-sensitive Markov decision processes framework, and Bäuerle and Ott [7] consider the value at risk of performance. In distributional RL settings, Bodnar et al. [11] incorporated various risk distortion metrics such as conditional value at risk CVaR), which enables robots to manage risk using a Deep RL control policy. They all assume, in general, that the safety function has some additive structure over time steps, as the cumulative reward is calculated. Hence, these methods are not appropriate for the purpose of guaranteeing safety at every time step; thus, the required level of safety is relatively low.

For safe RL approaches that require the agent to guarantee safety at every time step, we must provide agents with a certain prior knowledge. Eysenbach et al. [20] and Turchetta et al. [84] aimed to achieve safety by leveraging a library of reset controllers as prior knowledge, which are triggered and prevent further interactions with the environment when the agent is about to execute unsafe actions. However, such an approach can be used only when the agent is allowed to be reset in such cases where the agent is monitored by human during training in a laboratory. Hence, the approaches based on reset controllers cannot be used when agents are not allowed to start over from the beggining of the episode such as planetary rover or autonomous driving applications. In the problem settings where the agent needs to guarantee safety without *resetting*, [9] provided the agent with prior information of 1) what states are safe and 2) a part of the parameters of the state transition function under the name of safe model-based RL [21, 17, 45]. Thus, the resulting algorithm is well suited for such contexts as a drone learning how to hover under the condition that the system dynamics of the drone is partially unknown a priori. The parameters of a drone are not perfectly known a priori, but we have prior knowledge on what states are unsafe (e.g., a pitch angle of more than 50 degrees is unsafe). The above previous work focused on the uncertainty of the dynamics of the system (i.e., state transition function), and is not capable of dealing with the uncertainty of the safety associated with the environment.

Under the problem settings where agents are extremely discouraged to make even a single mistake during learning the environment (without resetting), several previous work assumed that safety function associated with the environment has some desirable structure under the name of *safe exploration* [10, 82, 78, 83, 87]. Specifically, Sui et al. [77] assumed that the safety of the environment has some regularity that can be captured by a certain kernel function; that is, the safety function value of a state is assumed to be similar to the one of neighboring states. Under this assumption, in a safety-constrained MDP, Wachi et al. [87] proposed an algorithm for maximizing the cumulative reward under safety constraints while learning the safety function structure with Gaussian processes. This previous work guaranteed safety with high probability, but there was no guarantee with regard to the optimality of the acquired policy. Given that the primary objective is still to maximize the cumulative reward, it is an open problem how to acquire the optimal policy while exploring an environment with a priori unknown reward and safety functions without resetting.

Validation of the safety of a policy.

On the other hands, no matter how carefully a software is designed, bugs and glitches could occur. Thus, efficient software testing is required for making software development more productive. Automated software testing is preferable because human software testing is expensive in terms of cost and efficiency. Artzi et al. [4] proposed a method for conducting random test case generation for web applications written by JavaScript. They aimed to find test suites with high coverage as well as sequences leading to programming errors. Marchetto and Tonella [46] generated a test suite of AJAX applications using a meta-heuristic algorithm. They ran the application to obtain a machine, and its state is the application's DOM-tree (Document object model) and its transitions (e.g., messages from the server and/or user input). Their goal was to generate a test suite with a maximum variety of event interactions. As a method for automated software testing, reinforcement learning has also been leveraged. For example, Bauersfeld and Vos [8] proposed a reinforcement learning based approach for finding bugs in Mac OSX software. They leveraged a Q-learning and used a manually defined Q-table, where a Q-value is learned for each state-action pair. Also, Harries et al. [28] formulated a software testing task as an Markov decision process and solved it using deep reinforcement learning called DRIFT. They regarded the tree-structured symbolic representation of the software as the state, and then modeled a generalizeable Q-function with Graph Neural Networks (GNNs). The above previous studies have typically addressed discrete and tabular MDP. Also, in the field of autonomous driving, several previous studies addressed how to test the software, but great deal of previous work [57, 81] has focused on test-case-generation related to image recognition. Hence, it is open problem of how to create test cases related to decision making of the autonomous driving vehicle in continuous state and action spaces. As software testing technologies, Nonnengart et al. [55] proposed an approach called CriSGen based on formal method for creating critical traffic scenarios in a automated and complete manner. Wang et al. [88] incorporated planning and sensing aspects of autnomous driving system and then proposed a method to create adversarial scenarios for LiDAR-based self-driving cars.

1.1 Contributions

In this thesis, we discuss how to design and validate an autonomous agent for safety-critical applications while leveraging RL paradigm with high performance.

Safe reinforcement learning with theoretical guarantee on near-optimality (Chapter 3). First, we consider *designing* safe and reliable software of the agent using reinforcement learning. We propose a safe reinforcement learning algorithm, called safe near-optimal MDP, SNO-MDP algorithm, for achieving a near-optimal cumulative reward while guaranteeing safety at every time step without resetting. Our algorithm learns a priori unknown environments while learning reward and safety function structures via Gaussian processes under regularity assumptions. We examine SNO-MDP by applying PAC-MDP analysis and prove that, with high probability, the acquired policy is near-optimal with respect to the cumulative reward while guaranteeing safety. Additionally, we build an openly-available test-bed called GP-SAFETY-GYM for synthetic experiments and empirically show that our algorithm performs better than existing methods.

Multi-agent adversarial reinforcement learning for policy validation (Chapter 4). Second, we consider *validating* the policy of the target agent using reinforcement learning with continuous state and action spaces. For the purpose of finding as many failure cases as possible before deployment, we propose a method for efficiently finding failure scenarios by training the (other) adversarial agents using multi-agent deep reinforcement learning such that the target agent fails. We see two advantages in this method. First, due to the nature of reinforcement learning, adversarial vehicles learn strategies by trial and error, which reduces human intervention and human resources. Next, reinforcement learning agents optimize their behavior by a reward function, which is completely different from the way humans create failure cases. Thus, it may create a failure case that is difficult for humans to find.

Software's Safety and RL. In this dissertation, we discuss how to leverage reinforcement learning techniques for reliable agent's policies. Comparing with conventional techniques, RL (being represented by deep RL) has recently achieved remarkable success. Unfortunately, however, the success of RL has not been sufficiently reflected in actual problems. In this dissertation, under the consideration that RL is one of the most promising approaches for optimize the policy of the agent, we propose two RL-based approaches for achieving reliable societies: one aims to train an agent safely using RL, and the other aims to validate the policy of the traget agent using multi-agent adversarial RL. We hope that the proposed approaches contribute to the realization of a safer society, where autonomous agents behave in a safer and more useful manner in the real environment.

1.2 Publications Relevant to this Dissertation

This thesis is based on the following peer-reviewed articles. The first article is for the policy optimization and the last two are for the policy validation. To be noted that the first article is under collaboration with Dr. Yanan Sui.

- Akifumi Wachi and Yanan Sui. Safe Reinforcement Learning in Constrained Markov Decision Processes. In International Conference on Machine Learning (ICML), 2020.
- Akifumi Wachi. Failure-Scenario Maker for Rule-Based Agent using Multi-agent Adversarial Reinforcement Learning and its Application to Autonomous Driving. In International Joint Conference on Artificial Intelligence (IJCAI), 2019.
- 和地 瞭良 (2020). 自動運転車に対する敵対的強化学習を用いた失敗ケース生成手法. 自動車技術会論文集, 51, 950-955.

Chapter 2

Background

2.1 Policy Optimization for Dynamical Systems

Dynamical systems are key basis of both control theory and reinforcement learning. Under the Markov property assumption, we now consider the dynamical system at discrete time step t. Let $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ denote the state and action at time t. Here, the state transition from $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$ by the action $\boldsymbol{a}_t$ can be described as

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t), \tag{2.1}$$

where the function f is a state transition function, which depends on the *i.i.d* transition noise $\boldsymbol{\omega}_t$ with $\mathbb{E}[\boldsymbol{\omega_t}] = 0$. When we deal with deterministic system without stochasticity (i.e., $\omega = 0$, the state transition function (2.1) can be represented as

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.2}$$

The objective is to obtain a policy $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$; that is, we aim to calculate the optimal action given the state $\boldsymbol{x}_t$ and time t. As an evaluation metric for the quality of the action, we define the reward function $r(\boldsymbol{s}, \boldsymbol{a})$. Note that, in control theory, "cost" function is usually employed instead of "reward" function. In summary, the optimization problem can be formulated as the one maximizing the (discounted) cumulative reward $J_\pi : \mathcal{S} \rightarrow \mathbb{R}$ as follows:

$$\begin{aligned} \max_{\boldsymbol{\pi}} \quad & J_{\boldsymbol{\pi}}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{\omega_t}}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)) \,\middle|\, \boldsymbol{s}_0 = \boldsymbol{s} \right] \\ \text{subject to} \quad & \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t), \boldsymbol{\omega}_t), \end{aligned}$$

where $\gamma \in [0, 1)$ is a discount factor.

2.2 Optimal Control

In the optimal control theory, the reward function r and transition model f are assumed to be *known a priori*. Intuitively, the objective of the optimal control theory is to get a policy π so that the agent with dynamics f could continue to take good state-action pairs that enjoys high rewards.

Policy optimization by approximate dynamic programming. A popular approach is approximate dynamic programming. In approximate dynamic programming, the objective function $J_\pi(\boldsymbol{s})$ for a given state $\boldsymbol{s}$ and policy π are modeled with a parametric function approximator. Policy improvement by approximate dynamic programming is conducted as follows. Let an approximation of J_π denote $\tilde{J}_\pi$, we aim to optimize a policy π_θ parameterized by θ. In summary,

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J_{\boldsymbol{\pi}}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{\omega}_t}\left[\gamma^T \tilde{J}_\pi(\boldsymbol{s}_T) + \sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t))\right] \\ \text{subject to} \quad & \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t), \boldsymbol{\omega}_t), \end{aligned}$$

Model predictive control. An alternative approach is model predictive control [13]. In model predictive control, the control input $\boldsymbol{a}_t$ is directly optimized over a finite horizon T, without parameterization. Policy is given by

$$\begin{aligned} \pi(\boldsymbol{s}) = \arg\min_{\boldsymbol{a}_0} \min_{\boldsymbol{a}_{1:T}} \mathbb{E}_{\boldsymbol{\omega}_t}\left[\gamma^T \tilde{J}_\pi(\boldsymbol{s}_T) + \sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \\ \text{subject to} \quad \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \pi_\theta(\boldsymbol{s}_t), \boldsymbol{\omega}_t), \end{aligned}$$

where $\boldsymbol{a}_{0:T}$ is an optimal control sequence from time $t = 0$ to T.

2.3 Reinforcement Learning

In reinforcement learning, the reward function r and/or the transition model f are assumed to be *unknown a priori*. Reinforcement learning algorithms can be classified into several metrics. The first metric is whether the model is explicitly considered. In the *model-based reinforcement learning*, we first learn a model of a system using the data and then solve the control problem using the learned model. On the other hand, in the *model-free reinforcement learning*, we directly optimize the policy by means of the sampled trajectories without explicitly modeling the system. Another metric is value-based RL vs. policy based RL. In the following, background algorithm necessary for understanding Chapter 3 and Chapter 4.

Value-based RL algorithms. Value iteration is one of the most fundamental algorithms in RL under the assumption that the transition is known. The optimal value function is given by

$$V^*(s) := \max_\pi V^\pi(s),$$

which is the largest value in terms of the long-term cumulative reward from a state s. A optimal policy π^* satisfies

$$V^{\pi^*} = V^*(s) \quad \forall s \in \mathcal{S}.$$

It is known that value iteration algorithm achieves the convergence guarantee. [79] Let n be the number of iteration. Then, we have

$$\lim_{n\to\infty} |\, V_n - V^* \,| = 0.$$

To be noted that the value function and Q-function satisfy $V^*(s) = \max_a Q^*(s,a)$.

When the state transition is unknown a priori, Q-learning is frequently used which leverages action-value function

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s = s_0, a = a_0 \right].$$

This Q-function is known to satisfy the following Bellman equation:

$$Q(s,a) = \mathbb{E}\left[r(s,a) + \gamma Q(s', a') \right].$$

While conventional Q-learning methods have obtained the optimal policy using Q-table, deep RL algorithms has been paid attention recently. In deep Q-networks (DQN), the action-value function Q^* for the optimal policy π^* is obtained by minimizing the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[(Q^*(s,a \mid \theta) - y)^2 \right],$$

where θ is a parameter of the network which is periodically updated. Also, y is defined as follows using the target Q-function, $\bar{Q}^*$

$$y = r + \gamma \max_{a'} \bar{Q}^*(s', a').$$

Policy-based RL. Policy gradient (PG) algorithms are popular in RL tasks. The key idea of PG is to directly optimize the parameters θ of policy π to maximize the expected cumulative reward by calculating the policy's gradient with regard to θ.

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a \mid s) Q^\pi(s, a)],$$

One of the main focus in PG algorithms is how to estimate Q^π. When we simply use a sample return $\sum_{\tau=t}^{T} \gamma^{i-t} r_i$, it is called REINFORCE [89] algorithm.

Deterministic policy gradient (DPG) algorithms are variants of PG algorithms that extend the PG framework to deterministic policy, $\boldsymbol{\mu}_\theta : \mathcal{S} \to \mathcal{A}$. In the DPG framework, the gradient of the objective function $J(\theta) = \mathbb{E}_{s \sim p^{\boldsymbol{\mu}}}[R(s, a)]$ is written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \mathcal{D}}[\nabla_\theta \boldsymbol{\mu}_\theta(s) \nabla_a Q^{\boldsymbol{\mu}}(s, a)|_{a = \boldsymbol{\mu}_\theta(s)}],$$

where $\rho^{\boldsymbol{\mu}}$ is the state distribution, and $\mathcal{D}$ is the replay buffer that contains the tuple, $(\boldsymbol{s}, a, r, \boldsymbol{s}')$.

Deep deterministic policy gradient (DDPG) approximates the deterministic policy $\boldsymbol{\mu}$ (i.e., actor) and critic $Q^{\boldsymbol{\mu}}$ (i.e., critic) using deep neural networks, which is a variant of actor-critic algorithms [37] as with asynchronous advantage actor-critic (A3C) and advantage actor-critic (A2C) algorithms (Mnih et al. [49]).

2.4 Exploration and Exploitation

Because the RL problems solve the problem with the unknown reward function and state transition function, the agent cannot obtain the optimal policy the beginning. Hence, the agent needs to learn the *a priori* unknown functions (i.e., get more knowledge on the function structure). This process is called "exploration." On the other hand, once the agent acquires precise reward and state transition functions, the agent could optimize the policy by maximizing the cumulative reward on the basis of the known reward and safety functions. This process is called "exploitation."

Intuitively, exploration-exploitation dilemma is exampled by the process to find a good restaurant. Here, exploration is to go to the restaurant for which you do not have information (i.e., price, taste), and exploitation is to go to your favorite restaurant. In order to identify the best restaurant, you need to explore new restaurants, but a new restaurant may be terrible. For the identification of the optimal solution, you must solve exploration-exploitation dilemma.

Exploration-exploitation dilemma. A fundamental problem in RL is called *exploration and exploitation dilemma*. If the agent spend too much time for exploration, the agent could not get much cumulative reward and the acquired policy is poor. If the agent tries to optimize policy with imprecise reward and state transition, the acquired policy is not consistent with the true environment. Therefore, the agent mush balance exploration and exploitation.

Safety in Exploration. A problem addressed in this dissertation is related to safety in exploration. Exploration in RL usually conducted by randomly sampling the next action (e.g., ϵ-greedy); hence, the agent would execute unsafe actions. In the exploration phase, the agent does not know the precise reward and/or state transition functions, so it would be more difficult to avoid unsafe actions than in exploitation phase.[1] Recently, several studies [77, 82, 87, 83] addressed "safe exploration" problems, which tries to guarantee or encourage safety during exploration phase as well as exploitation.

2.5 Sample Complexity and PAC-MDP

Over the decades, theoretical studies on RL have been conducted. A great deal of such previous work has focused on efficiency in terms of acquiring the cumulative reward. Representatives of such work are *probably approximately correct Markov decision process (PAC-MDP)* algorithms [12, 33, 74]. Algorithms with the PAC-MDP property enable an agent to learn a near-optimal behavior with a polynomial number of samples. The formal definition of PAC-MDP algorithm is given as follows:

Definition 1 (Sample Complexity, Kakade et al. [31]) For any fixed $\epsilon > 0$, the sample complexity is defined as the number of time-steps t such that the policy π satisfies

$$V^{\pi}(s_t) < V^{*}(s_t) - \epsilon.$$

.

Definition 2 (PAC-MDP, Strehl and Littman [75]) An algorithm is said to be a PAC-MDP algorithm if, for any $\epsilon > 0$ and $\delta \in [0, 1]$, its per-timestep computational complexity,

[1]If we desire to guarantee safety only in the exploitation phase, we can provide the agent some penalty (i.e., negative reward) for discourage it from making the same mistake.

space complexity, and the sample complexity of are less than some polynomial in the relevant quantities, with probability at least $1 - \delta$.

In addition, Kolter and Ng [36] and Araya et al. [3] proposed algorithms to obtain an ϵ-close solution to the Bayesian optimal policy. For example, Kolter and Ng [36] approximated the transition probability with some appropriate distribution (e.g., Dirichlet distribution) and aimed to maximize expected reward over the probabilistic distribution.

Chapter 3

Safe Reinforcement Learning in Constrained Markov Decision Processes

In many real applications, environmental hazards are first detected *in situ*. For example, a planetary rover exploring Mars does not obtain high-resolution images at the time of its launch. In usual cases, after landing on Mars, the rover takes close-up images or observes terrain data. Leveraging the acquired data, ground operators identify whether each position is safe. Hence, for fully automated operation, an agent must autonomously *explore* the environment and *guarantee* safety.

In most cases, however, guaranteeing safety (i.e., surviving) is *not* the primary objective. The optimal policy for ensuring safety is often extremely conservative (e.g., stay at the current position). Even though avoiding hazards is an essential requirement, the primary objective is nonetheless to obtain rewards (e.g., scientific gain).

As a framework to solve this problem, safe reinforcement learning (safe RL, Garcıa and Fernández [24]) has recently been noticed by the research community. The objective of safe RL is to maximize the cumulative reward while guaranteeing or encouraging safety. Especially in problem settings in which the reward and safety functions are *unknown a priori*, however, a great deal of previous work (e.g., Wachi et al. [87]) theoretically guarantees the satisfaction of the safety constraint, but the acquired policy is not necessarily near-optimal in terms of the cumulative reward. In this paper, we propose a safe RL algorithm that guarantees a near-optimal cumulative reward while guaranteeing the satisfaction of the safety constraint as well.

As the research community tries to apply RL algorithms to real-world systems, however, safety issues have been highlighted. RL algorithms inherently require an agent to explore unknown state-action pairs, and algorithms that are agnostic with respect to safety may execute unsafe actions without deliberateness. Hence, it is important to develop algorithms that guarantee safety even during training, at least with high probability.

Our contributions.

We propose a safe near-optimal MDP, SNO-MDP algorithm, for achieving a near-optimal cumulative reward while guaranteeing safety at every time step in a constrained MDP under the regularity assumption. This algorithm first explores the safety function and then optimizes the cumulative reward in the certified safe region. We further propose an algorithm called Early Stopping of Exploration of Safety (ES^2) to achieve faster convergence while maintaining probabilistic guarantees with respect to both safety and reward. We examine SNO-MDP by applying PAC-MDP analysis and prove that, with high probability, the acquired policy is near-optimal with respect to the cumulative reward while guaranteeing safety. We build an openly-available test-bed called GP-SAFETY-GYM for synthetic experiments.

Source-code: https://github.com/akifumi-wachi-4/safe_near_optimal_mdp

The safety and efficiency of SNO-MDP are then evaluated with two experiments: one in the GP-SAFETY-GYM synthetic environment, and the other using real Mars terrain data.

3.1 Problem Statement

A safety constrained MDP is defined as a tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, f, r, g, \gamma \rangle,$$

where $\mathcal{S}$ is a finite set of states $\{\boldsymbol{s}\}$, $\mathcal{A}$ is a finite set of actions $\{a\}$, $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a deterministic state transition function, $r : \mathcal{S} \rightarrow (0, R_{\max}]$ is a bounded reward function, $g : \mathcal{S} \rightarrow \mathbb{R}$ is a safety function, and $\gamma \in \mathbb{R}$ is a discount factor. We assume that both the reward function r and the safety function g are *not known a priori*. At every time step $t \in \mathbb{N}$, the agent must be in a "safe" state. More concretely, for a state $\boldsymbol{s}_t$, the safety function value $g(\boldsymbol{s}_t)$ must be above a threshold $h \in \mathbb{R}$; that is, the safety constraint is represented as

$$g(\boldsymbol{s}_t) \geq h.$$

A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ maps a state to an action. The value of a policy is evaluated according to the discounted cumulative reward under the safety constraint. Let $V_{\mathcal{M}}$ denote the value function in the MDP, $\mathcal{M}$. In summary, we represent our problem as follows:

$$\begin{aligned} \text{maximize:} \quad & V^{\pi}_{\mathcal{M}}(\boldsymbol{s}_t) = \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^{\tau} r(\boldsymbol{s}_{t+\tau}) \,\middle|\, \boldsymbol{s}_t \right] \\ \text{subject to:} \quad & g(\boldsymbol{s}_{t+\tau}) \geq h, \quad \forall \tau = [0, \infty]. \end{aligned}$$

Difficulties. In conventional safety-constrained RL algorithms, the safety function is assumed to be known a priori. The key difference lies in the fact that we need to explore a safety function that is unknown a priori while guaranteeing satisfaction of the safety constraint.

However, it is intractable to solve the above problem without further assumptions. First of all, without prior information on the state-and-action pairs known to be safe, an agent cannot take any viable action at the very beginning. Second, if the safety function does not exhibit any regularity, then the agent cannot infer the safety of decisions.

Assumptions. To overcome the difficulties mentioned above, we adopt two assumptions from Sui et al. [77] and Turchetta et al. [82]. For the first difficulty, we simply assume that the agent starts in an initial set of states, S_0, that is known a priori to be safe. Second, we assume regularity for the safety function. Formally speaking, we assume that the state space $\mathcal{S}$ is endowed with a positive definite kernel function, k^g, and that the safety function has a bounded norm in the associated reproducing kernel Hilbert space (RKHS, Schölkopf and Smola [63]). The kernel function, k^g is employed to capture the regularity of the safety function. Finally, we further assume that the safety function g is L-Lipschitz continuous with respect to some distance metric $d(\cdot, \cdot)$ on $\mathcal{S}$.

As with the safety function, we also assume that the reward function has a bounded norm in the associated RKHS, and that its regularity is captured by another positive definite kernel function, k^r.

The above assumptions allow us to characterize the reward and safety functions by using Gaussian processes (GPs, see Rasmussen [59]). By using the GP models, the values of r and g at unobserved states are predicted according to previously observed functions' values. An advantage of leveraging GPs is that we can obtain both optimistic and pessimistic measurements of the two functions by using the inferred means and variances. A GP is specified by its mean, $\mu(\boldsymbol{s})$, and covariance, $k(\boldsymbol{s}, \boldsymbol{s}')$. The reward and safety functions are thus modeled as

$$\begin{aligned} r(\boldsymbol{s}) &= \mathcal{GP}(\mu^r(\boldsymbol{s}), k^r(\boldsymbol{s}, \boldsymbol{s}')), \\ g(\boldsymbol{s}) &= \mathcal{GP}(\mu^g(\boldsymbol{s}), k^g(\boldsymbol{s}, \boldsymbol{s}')). \end{aligned}$$

Without loss of generality, let $\mu(\boldsymbol{s}) = 0$ for all $\boldsymbol{s} \in \mathcal{S}$. For the reward and safety functions, we respectively model the observation noise as

$$\begin{aligned} y^r &= r(\boldsymbol{s}) + n^r, \\ y^g &= g(\boldsymbol{s}) + n^g, \end{aligned}$$

where $n^r \sim \mathcal{N}(0, \sigma_r^2)$ and $n^g \sim \mathcal{N}(0, \sigma_g^2)$. The posteriors over r and g are computed on the basis of t observations at states $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_t\}$. Then, for both the reward and safety functions, the posterior mean, variance, and covariance are respectively represented as

$$\begin{aligned} \boldsymbol{\mu}_t(\boldsymbol{s}) &= \boldsymbol{k}_t^\top(\boldsymbol{s})(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}_t, \\ \boldsymbol{\sigma}_t(\boldsymbol{s}) &= \boldsymbol{k}_t(\boldsymbol{s}, \boldsymbol{s}), \\ \boldsymbol{k}_t(\boldsymbol{s}, \boldsymbol{s}') &= \boldsymbol{k}(\boldsymbol{s}, \boldsymbol{s}') - \boldsymbol{k}_t^\top(\boldsymbol{s})(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(\boldsymbol{s}'), \end{aligned}$$

where $\boldsymbol{k}_t(\boldsymbol{s}) = [k(\boldsymbol{s}_1, \boldsymbol{s}), \ldots, k(\boldsymbol{s}_t, \boldsymbol{s})]^\top$. Finally, $\boldsymbol{K}_t$ is called positive definite kernel matrix and given by

$$\boldsymbol{K} = \begin{pmatrix} k(s_1, s_1) & k(s_1, s_2) & \ldots & k(s_1, s_n) \\ k(s_2, s_1) & k(s_2, s_2) & \ldots & k(s_2, s_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(s_n, s_1) & k(s_n, s_2) & \ldots & k(s_n, s_n) \end{pmatrix}.$$

3.2 Background

We define two kinds of predicted safe spaces inferred by a GP as in Turchetta et al. [83]. First, we consider a *pessimistic* safe space, which contains states identified as safe with a greater probability than a pre-defined confidence level. Second, we derive an *optimistic* safe space that includes all states that may be safe with even a small probability.

Predicted pessimistic safe space. We use the notion of a safe space in Turchetta et al. [82] as a *predicted pessimistic safe space*. For the probabilistic safety guarantee, two sets are defined. The first set, S_t^-, simply contains the states that satisfy the safety constraint with high probability. The second one, $\mathcal{X}_t^-$, additionally considers the ability to reach states in S_t^- (i.e., reachability) and the ability to return to the previously identified safe set, $\mathcal{X}_{t-1}^-$ (i.e., returnability). The algorithm probabilistically guarantees safety by allowing the agent to visit only states in $\mathcal{X}_t^-$.

Safety is evaluated in terms of the confidence interval inferred by the GP, which is represented as

$$Q_t(\boldsymbol{s}) := [\mu^g_{t-1}(\boldsymbol{s}) \pm \beta_t^{1/2} \sigma^g_{t-1}(\boldsymbol{s})],$$

where $\beta_t \in \mathbb{R}$ is a scaling factor for the required level of safety. We consider the intersection of Q_t up to iteration t, which is defined as

$$C_t(\boldsymbol{s}) = Q_t(\boldsymbol{s}) \cap C_{t-1}(\boldsymbol{s}),$$

where $C_0(\boldsymbol{s}) = [h, \infty]$ for all $\boldsymbol{s} \in S_0$. The lower and upper bounds on $C_t(\boldsymbol{s})$ are denoted by $l_t(\boldsymbol{s}) := \min C_t(\boldsymbol{s})$ and $u_t(\boldsymbol{s}) := \max C_t(\boldsymbol{s})$, respectively.

The first set S_t^- contains states such that the safety constraint is satisfied with high probability. It is formulated using the lower bound of the safety function, l and the Lipshitz constant, L, as follows:

$$S_t^- = \{\boldsymbol{s} \in \mathcal{S} \mid \exists \boldsymbol{s}' \in \mathcal{X}_{t-1}^- : l_t(\boldsymbol{s}') - L \cdot d(\boldsymbol{s}, \boldsymbol{s}') \geq h\}.$$

Next, the reachable and returnable sets are considered. Even though a state is in S_t^-, it might be surrounded by unsafe states. Given a set X, the states reachable from X in one step are given by

$$R_{\text{reach}}(X) = X \cup \{\boldsymbol{s} \in \mathcal{S} \mid \exists \boldsymbol{s}' \in X, a \in \mathcal{A} : \boldsymbol{s} = f(\boldsymbol{s}', a)\}.$$

Even after arriving at a state with reachability, the agent may *not* be able to move to another state because of a lack of safe actions. Hence, before moving to a state $\boldsymbol{s}$, we consider whether or not there is at least one viable path from $\boldsymbol{s}$. The set of states from which the agent can return to a set $\bar{X}$ through another set of states X in one step is given by

$$R_{\text{ret}}(X, \bar{X}) = \bar{X} \cup \{\boldsymbol{s} \in X \mid \exists a \in \mathcal{A} : f(\boldsymbol{s}, a) \in \bar{X}\}.$$

Thus, an n-step returnability operator is given by

$$R^n_{\text{ret}}(X, \bar{X}) = R_{\text{ret}}(X, R^{n-1}_{\text{ret}}(X, \bar{X})), \text{with } R^1_{\text{ret}}(X, \bar{X}) = R_{\text{ret}}(X, \bar{X}).$$

Finally, the set containing all the states that can reach $\bar{X}$ along an arbitrary long path in X is defined as

$$\bar{R}_{\text{ret}}(X, \bar{X}) = \lim_{n \to \infty} R^n_{\text{ret}}(X, \bar{X}).$$

Finally, the desired *pessimistic* safe space, $\mathcal{X}_t^-$ is a subset of S_t^- and also satisfies the reachability and returnability constraints; that is,

$$\mathcal{X}_t^- = \{\boldsymbol{s} \in S_t^- \mid \boldsymbol{s} \in R_{\text{reach}}(\mathcal{X}_{t-1}^-) \cap \bar{R}_{\text{ret}}(S_t^-, \mathcal{X}_{t-1}^-)\}.$$

Predicted optimistic safe space. As defined in Wachi et al. [87] and Turchetta et al. [83], an *optimistic* safe space has rich information for inferring the safety function. Let $\mathcal{X}_t^+$ denote the predicted optimistic safe space. Similarly to $\mathcal{X}_t^-$, the *optimistic* safe space, $\mathcal{X}_t^+$, is defined as

$$\mathcal{X}_t^+ = \{\boldsymbol{s} \in S_t^+ \mid \boldsymbol{s} \in R_{\text{reach}}(\mathcal{X}_{t-1}^+) \cap \bar{R}_{\text{ret}}(S_t^+, \mathcal{X}_{t-1}^+)\},$$

where S_t^+ is the set of states that may satisfy the safety constraint, which is written as

$$S_t^+ = \{\boldsymbol{s} \in \mathcal{S} \mid \exists \boldsymbol{s}' \in \mathcal{X}_{t-1}^+ : u_t(\boldsymbol{s}') - L \cdot d(\boldsymbol{s}, \boldsymbol{s}') \geq h\}.$$

Intuitively, $\mathcal{X}_t^+$ contains all states that may turn out to be safe even if the probability is low. In other words, $\mathcal{S} \setminus \mathcal{X}_t^+$ contains states that are unsafe with high probability.

Confidence interval. The correctness of $\mathcal{X}_t^+$ and $\mathcal{X}_t^-$ depends on the accuracy of the confidence interval inferred by the GP. The *conservativeness* can be tuned by using the parameter β, and the choice of this parameter was well-studied in Srinivas et al. [72] and Chowdhury and Gopalan [16]. In the rest of this paper, we set the parameter to

$$\beta_t = B^g + \sigma_g \sqrt{2(\Gamma_{t-1}^g + 1 + \log(1/\Delta^g))},$$

where B^g is a bound on the RKHS norm of g, Δ^g is the allowed failure probability, and the observation noise is σ_g-sub-Gaussian. Also, Γ^g quantifies the effective degrees of freedom associated with the kernel function, which represents the maximal mutual information that can be obtained about the GP prior.

Under the above definitions and assumptions, we have guarantees regarding the correctness of the confidence intervals. Hence, we have the following lemmas for both the reward and safety functions.

Lemma 1. *Assume that* $\|g\|_k^2 \leq B^g$ *and* $n_t^g \leq \sigma_g$, $\forall t \geq 1$. *If*

$$\beta_t = B^g + \sigma_g \sqrt{2(\Gamma_{t-1}^g + 1 + \log(1/\Delta^g))},$$

then the following inequality holds for all $t \geq 1$ *with probability at least* $1 - \Delta^r$.

$$|g(\boldsymbol{s}) - \mu^g_{t-1}(\boldsymbol{s})| \leq \beta_t^{1/2} \sigma^g_{t-1}(\boldsymbol{s}).$$

Lemma 2. *Assume that* $\|r\|_k^2 \leq B^r$ *and* $n^r_t \leq \sigma_r$, $\forall t \geq 1$. *If*

$$\alpha_t = B^r + \sigma_r \sqrt{2(\Gamma^r_{t-1} + 1 + \log(1/\Delta^r))},$$

then the following inequality holds for all $t \geq 1$ *with probability at least* $1 - \Delta^r$.

$$|r(\boldsymbol{s}) - \mu^r_{t-1}(\boldsymbol{s})| \leq \alpha_t^{1/2} \sigma^r_{t-1}(\boldsymbol{s}).$$

Proof. These lemmas follow from Theorem 2 in Chowdhury and Gopalan [16]. □

Optimal solution. Here, we define the optimal policy in our problem setting. Under the optimal policy, π^*, the value function, $V_\mathcal{M}$, satisfies the following Bellman equation:

$$V^*_\mathcal{M}(\boldsymbol{s}_t) = \max_{\boldsymbol{s}_{t+1} \in \bar{R}_{\epsilon_g}(S_0)} \left[\, r(\boldsymbol{s}_{t+1}) + \gamma V^*_\mathcal{M}(\boldsymbol{s}_{t+1}) \,\right],$$

where $\bar{R}_{\epsilon_g}(S_0)$ is the largest set that can be safely learned up to ϵ_g accuracy (for a formal definition, see Appendix A or Turchetta et al. [82]). In our problem setting, in which the reward and safety functions are unknown a priori, the above Bellman equation cannot be solved directly. Our ultimate objective is to obtain a policy whose value is close to $\mathcal{V}^*_\mathcal{M}$ while guaranteeing satisfaction of the safety constraint.

3.3 Algorithm

We now introduce our proposed algorithm, SNO-MDP, for achieving a near-optimal policy with respect to the cumulative reward while guaranteeing safety.

We first give an overview of SNO-MDP, which is outlined as Algorithm 1. We extend a stepwise approach in Sui et al. [78] from state-less to stateful settings. Basically, our algorithm consists of two steps. In the first step, the agent expands the pessimistic safe region while guaranteeing safety (lines 2–17). Next, it explores and exploits the reward in the safe region certified in the first step (lines 18–23). The reason for this stepwise approach is that we can neglect uncertainty related to the a priori unknown safety function once the safe region is fixed.

Algorithm 1 SNO-MDP **with** ES^2

Input: states $\mathcal{S}$, actions $\mathcal{A}$, transition function f, kernel k^r for reward, kernel k^g for safety, GP prior for reward, GP prior for safety, safety threshold h, discount factor γ, Lipschitz constant L, initial safe space S_0.

```
1: $C_0(\boldsymbol{s}) \leftarrow [h, \infty)$ for all $\boldsymbol{s} \in S_0$
2: // Exploration of safety
3: loop
4:     $S_t^- \leftarrow \{\boldsymbol{s} \in \mathcal{S} \mid \exists \boldsymbol{s}' \in \mathcal{X}_{t-1}^- : l_t(\boldsymbol{s}') - L \cdot d(\boldsymbol{s}, \boldsymbol{s}') \geq h\}$
5:     $S_t^+ \leftarrow \{\boldsymbol{s} \in \mathcal{S} \mid \exists \boldsymbol{s}' \in \mathcal{X}_{t-1}^+ : u_t(\boldsymbol{s}') - L \cdot d(\boldsymbol{s}, \boldsymbol{s}') \geq h\}$
6:     $\mathcal{X}_t^- \leftarrow \{\boldsymbol{s} \in S_t^- \mid \boldsymbol{s} \in R_{\text{reach}}(\mathcal{X}_{t-1}^-) \cap \bar{R}_{\text{ret}}(S_t^-, \mathcal{X}_{t-1}^-)\}$
7:     $\mathcal{X}_t^+ \leftarrow \{\boldsymbol{s} \in S_t^+ \mid \boldsymbol{s} \in R_{\text{reach}}(\mathcal{X}_{t-1}^+) \cap \bar{R}_{\text{ret}}(S_t^+, \mathcal{X}_{t-1}^+)\}$
8:     $G_t \leftarrow \{s \in \mathcal{X}_t^- \mid e_t(s) > 0\}$
9:     $\xi \leftarrow \arg\max_{\boldsymbol{s} \in G_t} w_t(\boldsymbol{s})$
10:    Update GPs for both reward and safety on way to $\xi$
11:    $t \leftarrow t + T_{\boldsymbol{s}_{t-1} \to \xi}$ and $\boldsymbol{s}_t \leftarrow \xi$
12:    // ES$^2$ algorithm
13:    $\mathcal{Y}_t \leftarrow \{\boldsymbol{s}' \in \mathcal{S}^+ \mid \forall \boldsymbol{s} \in \mathcal{X}_t^- : \boldsymbol{s}' = f(\boldsymbol{s}, \pi_y^*(a \mid \boldsymbol{s}))\}$
14:    if $\mathcal{Y}_t \subseteq \mathcal{X}_t^-$ then break
15:    // Typical termination condition
16:    if $\max_{\boldsymbol{s} \in G_t} w_t(s) < \epsilon_g$ then break
17: end loop
18: // Exploration and exploitation of reward
19: loop
20:    $U_t \leftarrow \mu_t^r + \alpha_{t+1} \cdot \sigma_t^r$
21:    $J_{\mathcal{Y}}^*(\boldsymbol{s}_t) \leftarrow \max_{\boldsymbol{s}_{t+1} \in \mathcal{Y}_t} \left[U_t(\boldsymbol{s}_{t+1}) + \gamma J_{\mathcal{Y}}^*(\boldsymbol{s}_{t+1})\right]$
22:    $\boldsymbol{s}_{t+1} \leftarrow \arg\max_{\boldsymbol{s}_{t+1} \in \mathcal{Y}_t} J_{\mathcal{Y}}^*(\boldsymbol{s}_t)$
23: end loop
```

However, a pure stepwise approach does not stop exploring the safe region until the convergence of the GP confidence interval (lines 15−16). This formulation often requires the agent to execute a great number of actions for exploring safety. Hence, to achieve near-optimality while executing a smaller number of actions, we also propose the ES^2 algorithm.[1] This algorithm checks whether the current safe region is sufficient for achieving near-optimality (lines 12−14), which maintains the theoretical guarantee with respect to both the satisfaction of the safety constraint and the near-optimality of the cumulative reward. We further propose a practical ES^2 algorithm, called P-ES^2, with better empirical performance, although it does not provide a theoretical guarantee in terms of the near-optimality of the cumulative reward.

[1]Both ES^2 and P-ES^2 do *not* affect the agent's safety.

3.3.1 Exploration of Safety (Step 1)

First, we consider how to explore the safety function. As a scheme to expand the safe region, we consider "expanders" as in Sui et al. [77] and Turchetta et al. [82]. Expanders are states that may expand the predicted safe region, which is defined as

$$G_t = \{s \in \mathcal{X}_t^- \mid e_t(s) > 0\}$$

In the above equation, we have

$$e_t(s) = |s' \in \mathcal{S} \setminus S_t^- \mid u_t(s) - Ld(s, s') \geq h|.$$

The *efficiency* of expanding the safe region is measured by the width of the safety function's confidence interval, defined as

$$w_t(\boldsymbol{s}) = u_t(s) - l_t(s).$$

The agent safely and efficiently expands the safe region by sampling the state with the maximum value of w among the expanders, G_t. Hence, the agent sets the temporal goal according to

$$\xi = \arg\max_{\boldsymbol{s} \in G_t} w_t(\boldsymbol{s}).$$

Then, within the predicted safe space $\mathcal{X}_t^-$, it chooses a path to get to ξ from the current state $\boldsymbol{s}_{t-1}$ so as to minimize the cost (e.g., the path length). In our experiment, we simply minimized the path length. By defining the cost as related to w (e.g., $1/w$), however, the agent could explore safety more actively on the way to ξ.

The previous work [77, 82, 78] terminated safety exploration when the desired accuracy was achieved for every state in G_t; that is,

$$\max_{s \in G_t} w_t(\boldsymbol{s}) \leq \epsilon_g. \tag{3.1}$$

Unfortunately, this termination condition often requires a great number of iterations. For the purpose of maximizing the cumulative reward, it often leads to the loss of reward. Therefore, in Section 3.3.3, we propose the ES^2 algorithm to improve this point.

3.3.2 Exploration and Exploitation of Reward (Step 2)

Once expansion of the safe region is completed, the agent guarantees safety as long as it is in $\mathcal{X}^-$ and does not have to expand the safe region anymore. Hence, all we have to do is

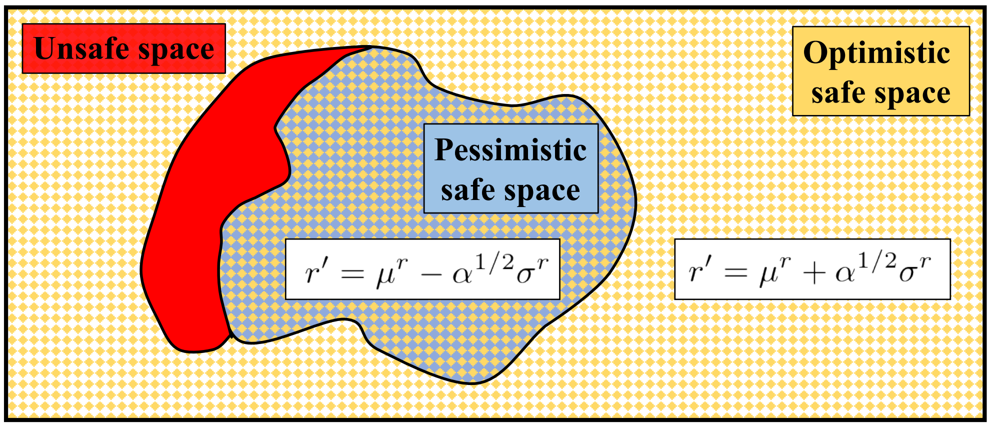

Unsafe space
Optimistic safe space
Pessimistic safe space
$r' = \mu^r - \alpha^{1/2}\sigma^r$
$r' = \mu^r + \alpha^{1/2}\sigma^r$

Fig. 3.1 Illustration of $\mathcal{M}_y$, used in the ES^2 algorithm. The yellow and blue regions represent $\mathcal{X}_t^+$ and $\mathcal{X}_t^-$, respectively. The red region (i.e., $\mathcal{X} \setminus \mathcal{X}_t^+$) is unsafe with high probability.

optimize the cumulative reward in $\mathcal{X}^-$. As such, a simple approach is to follow the *optimism in the face of uncertainty* principle as in Strehl and Littman [76] and Auer and Ortner [6], then to consider the "exploration bonus" represented by R-MAX [12] and Bayesian Exploration Bonus (BEB, Kolter and Ng [36]).

Specifically, in accordance with Lemma 2, we optimize the policy by *optimistically* measuring the reward with the (probabilistic) upper confidence bound,

$$U_t(\boldsymbol{s}) := \mu_t^r(\boldsymbol{s}) + \alpha_{t+1}^{1/2} \cdot \sigma_t^r(\boldsymbol{s}).$$

In this reward setting, the second term on the right-hand side corresponds to the exploration bonus. For balancing the exploration and exploitation in terms of reward, we solve the following Bellman equation:

$$J_{\mathcal{X}}^*(\boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) = \max_{s_{t+1} \in \mathcal{X}_{t^*}^-} \left[U_t(\boldsymbol{s}_{t+1}) + \gamma J_{\mathcal{X}}^*(\boldsymbol{s}_{t+1}, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) \right],$$

where $\boldsymbol{b}^r = (\mu^r, \sigma^r)$ and $\boldsymbol{b}^g = (\mu^g, \sigma^g)$ are the beliefs over reward and safety, respectively. Also, t^* is the time step when the termination condition (3.1) is satisfied. Note that $\boldsymbol{b}^r$ and $\boldsymbol{b}^g$ are not updated; hence, we can solve the above equation with standard algorithms (e.g., value iteration).

3.3.3 Early Stopping of Exploration of Safety

We have proposed a stepwise approach for exploring and optimizing the constrained MDP. In the first step when the safe region is expanded, however, the existing safe exploration algorithms [77, 82, 78] continue exploring the state space until convergence of the confidence interval, w, which generally leads to a large number of iterations. Our primary objective is to maximize the cumulative reward; hence, we should stop exploring safety if further exploration will not lead to maximizing the cumulative reward.

While exploring the safety function, we check whether the step can be migrated. As such, we consider the following additional MDP,

$$\mathcal{M}_y = \langle \mathcal{X}^+, \mathcal{A}, f, r', g, \gamma \rangle.$$

As shown in Figure 3.1, the differences from the original MDP, $\mathcal{M}$, lie in the state space and the reward function. The state space of $\mathcal{M}_y$ is defined as the optimistic safe space (i.e., $\mathcal{X}^+$), while the reward function is defined as follows:

$$r' := \begin{cases} \mu^r + \alpha^{1/2}\sigma^r & \text{if } \boldsymbol{s} \in \mathcal{X}_t^+ \setminus \mathcal{X}_t^-, \\ \mu^r - \alpha^{1/2}\sigma^r & \text{if } \boldsymbol{s} \in \mathcal{X}_t^-. \end{cases} \tag{3.2}$$

In the pessimistic safe space, the reward is defined as the lower bound; otherwise, it is defined as the upper bound. This definition of the reward function encourages the agent to explore outside the predicted safe space, $\mathcal{X}_t^-$. Using the new MDP above, we consider the set of states that the agent will visit at the next time step, defined as

$$\mathcal{Y}_t = \{\boldsymbol{s}' \in \mathcal{S}^+ \mid \forall \boldsymbol{s} \in \mathcal{X}_t^- : \boldsymbol{s}' = f(\boldsymbol{s}, \pi_y^*(a \mid \boldsymbol{s}))\},$$

where π_y^* is the optimal policy for $\mathcal{M}_y$, obtained by maximizing the following value function:

$$V_{\mathcal{M}_y}(\boldsymbol{s}_t) = \max_{s_{t+1} \in \mathcal{X}_t^+} [\, r'(\boldsymbol{s}_{t+1}) + \gamma V_{\mathcal{M}_y}(\boldsymbol{s}_{t+1}) \,]. \tag{3.3}$$

Finally, we stop exploring the safety function if the following equation holds:

$$\mathcal{Y}_t \subseteq \mathcal{X}_t^-. \tag{3.4}$$

Intuitively, we stop expanding the safe space if the direction of the optimal policy for $\mathcal{M}_y$ heads for the inside of $\mathcal{X}_t^-$. If the agent tries to stay in $\mathcal{X}_t^-$ even under the condition that the reward is defined as in (3.2), then we do not have to expand the safe region anymore.

When the ES^2 algorithm confirms satisfaction of the above condition, we move on to the next step and then optimize the cumulative reward in $\mathcal{Y}_t$; that is,

$$J_{\mathcal{Y}}^*(\boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) = \max_{s_{t+1} \in \mathcal{Y}_t} \left[U_t(\boldsymbol{s}_{t+1}) + \gamma J_{\mathcal{Y}}^*(\boldsymbol{s}_{t+1}, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) \right].$$

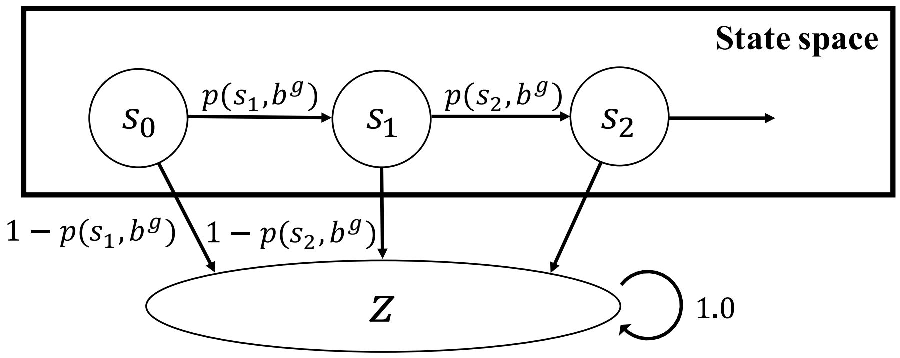


Fig. 3.2 Illustration of $\mathcal{M}_z$. This MDP is characterized by the virtual state z and the virtual transition probability P^z.

3.3.4 More Practical Early Stopping Algorithm

As we will prove in Section 3.4, the ES^2 algorithm provides us with a theoretical guarantee with respect to the cumulative reward. Unfortunately, this guarantee is achieved at the expense of empirical performance. The issue with the pure ES^2 algorithm lies in the state constraint in (3.3); that is, the value function is calculated under the assumption that all the states in the *optimistic* safe space, $\mathcal{X}^+$, will be identified as safe. This assumption is necessary for the theoretical guarantee, but, in practice, it would be more reasonable to measure the probability of a state being identified as safe. Because the safety function is inferred as a Gaussian distribution for each state, an example of such a probability is a complementary error function; that is, we define the following probability,

$$\begin{aligned} p(\boldsymbol{s}, \boldsymbol{b}^g) &= \Pr\left[g(\boldsymbol{s}) \geq h \mid \boldsymbol{b}^g\right] \\ &\approx 1 - \frac{1}{2}\mathrm{erfc}\left(\frac{\mu^g(\boldsymbol{s}) - h}{\sqrt{2}\sigma^g(\boldsymbol{s})}\right). \end{aligned}$$

Here, we introduce a new *virtual* state, $\boldsymbol{z}$. Concretely, for $\boldsymbol{z}$, the reward and transition probability are defined as $r(\boldsymbol{z}) = 0$ and $P(\boldsymbol{z} \mid \boldsymbol{z}, a, \boldsymbol{b}^g) = 1$ for all a and $\boldsymbol{b}^g$, respectively. Hence, using $\boldsymbol{z}$, we define a virtual transition probability $P^z_{\boldsymbol{x}} = \Pr[\boldsymbol{x} \mid \boldsymbol{s}_t, a_t, \boldsymbol{b}^g_t]$ as follows:

$$P^z_{\boldsymbol{x}} := \begin{cases} p(\boldsymbol{s}_{t+1}, \boldsymbol{b}^g_t) & \text{if } \boldsymbol{x} = \boldsymbol{s}_{t+1}, \\ 1 - p(\boldsymbol{s}_{t+1}, \boldsymbol{b}^g_t) & \text{if } \boldsymbol{x} = \boldsymbol{z}. \end{cases}$$

Hence, by introducing z and P^z, we define the following MDP with the smooth, continuous transition probability:

$$\mathcal{M}_z = \langle \mathcal{X}^+ \cup \{z\}, \mathcal{A}, P^z, r', g, \gamma \rangle.$$

Figure 3.2 shows a conceptual image of this MDP. Intuitively, in $\mathcal{M}_z$, the agent optimizes the policy under the virtual condition that a state-action pair may lead to the extremely undesirable state, z, with probability $1-p$. Then, we solve the following equation instead of solving (3.3):

$$V_{\mathcal{M}_z}(\boldsymbol{s}_t) = \max_{s_{t+1} \in \mathcal{X}_t^+} [P^z_{\boldsymbol{s}_{t+1}} \cdot \{r'(\boldsymbol{s}_{t+1}) + \gamma V_{\mathcal{M}_z}(\boldsymbol{s}_{t+1})\}].$$

For this equation, we used $r(\boldsymbol{z}) = 0$ and $V(\boldsymbol{z}) = 0$. For the optimal policy π^*_z obtained by solving the above equation, we stop exploring the safety function if the following equation holds:

$$\mathcal{Z}_t := \{\boldsymbol{s}' \in \mathcal{S}^+ \mid \forall \boldsymbol{s} \in \mathcal{X}_t^- : \boldsymbol{s}' = f(\boldsymbol{s}, \pi^*_z(a \,|\, \boldsymbol{s}))\} \subseteq \mathcal{X}_t^-.$$

Then, we optimize the cumulative reward in $\mathcal{Z}_t$ by solving the following equation:

$$J^*_{\mathcal{Z}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) = \max_{s_{t+1} \in \mathcal{Z}_t} \big[U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Z}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)\big].$$

3.4 Theoretical Results

We now provide theoretical guarantees on the safety and near-optimality of our proposed algorithm. Theorem 1 is associated with the safe expansion stage (i.e., step 1), which guarantees safety and convergence to the safe region. Theorem 2 ensures convergence toward the near-optimal cumulative reward. Theorem 3 ensures that SNO-MDP still achieves the near-optimal cumulative reward even when the ES^2 algorithm is used. In the rest of this paper, let

$$V_{\max} = \frac{R_{\max}}{1-\gamma}.$$

Also, let $D : \mathcal{M} \to \mathbb{R}$ be a diameter of an MDP, defined as

$$D(\mathcal{M}) = \min_{\pi} \max_{s^1 \neq s^2} T^{\pi}_{s^1 \to s^2},$$

where $T^{\pi}_{s^1 \to s^2}$ is the expected number of time steps that policy π takes to move from s^1 to s^2.

3.4.1 Safety Guarantee and Completeness

We first present a theorem related to the safety guarantee and completeness.

Theorem 1. *Assume that the safety function g satisfies $\|g\|^2_k \leq B^g$ and is L-Lipschitz continuous. Also, assume that $S_0 \neq \emptyset$ and $g(\boldsymbol{s}) \geq h$ for all $\boldsymbol{s} \in S_0$. Fix any $\epsilon_g > 0$ and $\Delta^g \in (0,1)$. Suppose that we conduct the stage of "exploration of safety" with the noise n^g_t*

being σ_g-sub-Gaussian, and that

$$\beta_t = B^g + \sigma_g \sqrt{2(\Gamma^g_{t-1} + 1 + \log(1/\Delta^g))}$$

until $\max_{\boldsymbol{s} \in G_t} w_t(\boldsymbol{s}) < \epsilon_g$ *is achieved. Finally, let t^* be the smallest integer satisfying*

$$\frac{t^*}{\beta_{t^*} \Gamma^g_{t^*}} \geq \frac{C_g |\bar{R}_0(S_0)|}{\epsilon_g^2} D(\mathcal{M}),$$

with $C_g = 8/\log(1 + \sigma_g^{-2})$. Then, the following statements jointly hold with probability at least $1 - \Delta^g$:

- $\forall t \geq 1$, $g(\boldsymbol{s}_t) \geq h$,
- $\exists t_0 \leq t^*$, $\bar{R}_{\epsilon_g}(S_0) \subseteq \mathcal{X}^-_{t_0} \subseteq \bar{R}_0(S_0)$.

A proof is presented in the supplemental material. Theorem 1 guarantees that SNO-MDP is safe in the stage of *exploration of safety* (i.e., step 1), as well as in the stage of *optimization of reward* (i.e., step 2), with high probability. In addition, after a sufficiently large number of time steps, $\mathcal{X}^-$ is guaranteed to be a super-set of $\bar{R}_{\epsilon_g}(S_0)$.

3.4.2 Near-Optimality

We next present a theorem on the near-optimality with respect to the cumulative reward.

Theorem 2. *Assume that the reward function r satisfies $\|r\|_k^2 \leq B^r$, and that the noise is σ_r-sub-Gaussian. Let π_t denote the policy followed by* SNO-MDP *at time t, and let $\boldsymbol{s}_t$ and $\boldsymbol{b}^r_t, \boldsymbol{b}^g_t$ be the corresponding state and beliefs, respectively. Let t^* be the smallest integer satisfying*

$$\frac{t^*}{\beta_{t^*} \Gamma^g_{t^*}} \geq \frac{C_g |\bar{R}_0(S_0)|}{\epsilon_g^2} D(\mathcal{M}),$$

and fix any $\Delta^r \in (0,1)$. Finally, set $\alpha_t = B^r + \sigma_r \sqrt{2(\Gamma^r_{t-1} + 1 + \log(1/\Delta^r))}$ and

$$\epsilon_V^* = V_{\max} \cdot (\Delta^g + \Sigma^r_{t^*}/R_{\max}),$$

with $\Sigma^r_{t^*} = \frac{1}{2}\sqrt{\frac{C_r \alpha_{t^*} \Gamma^r_{t^*}}{t^*}}$. Then, with high probability,

$$V^{\pi_t}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t) - \epsilon_V^*$$

— i.e., the algorithm is ϵ_V^*-close to the optimal policy — for all but t^* time steps, while guaranteeing safety with probability at least $1 - \Delta^g$.

A detailed proof of Theorem 2 is presented in the supplemental material. The proof is based on the following idea. After the agent fully explores the safe space, $\mathcal{X}^-$ satisfies $\bar{R}_{\epsilon_g}(S_0) \leq \mathcal{X}^- \leq \bar{R}_0(S_0)$, and states in $\mathcal{X}^-$ are safe with high probability. Once $\mathcal{X}^-$ converges, the probability of leaving the "known" safe space is small; hence, Theorem 2 follows by adapting standard arguments from previous PAC-MDP results. The key condition that allows us to prove the near-optimality of SNO-MDP is that, at every time step, the agent is optimistic with respect to the reward, and this optimism decays given a sufficient number of samples. By optimizing the cumulative reward in $\mathcal{X}^-$ according to the optimism in the face of uncertainty principle, the acquired policy is ϵ_V^*-close to the optimal policy in the original safety-constrained MDP.

Finally, we present a theoretical result related to the ES^2 algorithm. Specifically, we prove that ES^2 maintains the near-optimality of SNO-MDP.

Theorem 3. *Assume that the reward function r satisfies $\|r\|_k^2 \leq B^r$, and that the noise is σ_r-sub-Gaussian. Let π_t denote the policy followed by* SNO-MDP *with the* ES^2 *algorithm at time t, and let $\boldsymbol{s}_t$ and $\boldsymbol{b}_t^r, \boldsymbol{b}_t^g$ be the corresponding state and beliefs, respectively. Let $\tilde{t}$ be the smallest integer for which (3.4) holds, and fix any $\Delta^r \in (0, 1)$. Finally, set $\alpha_t = B^r + \sigma_r \sqrt{2(\Gamma_{t-1}^r + 1 + \log(1/\Delta^r))}$ and*

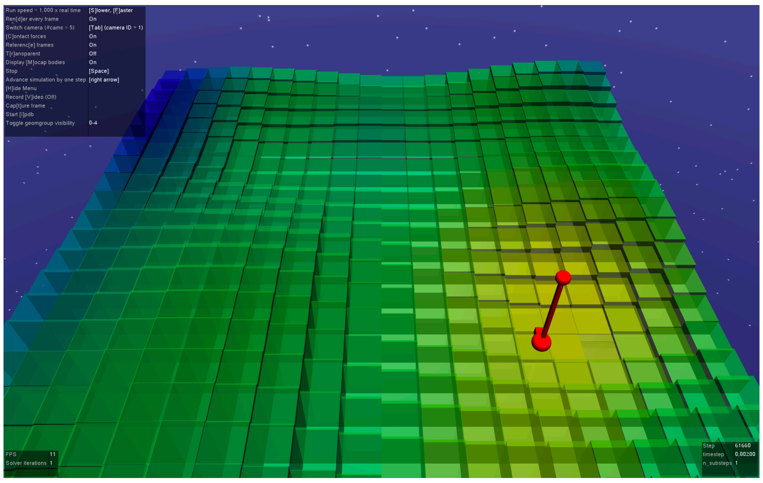
$$\tilde{\epsilon}_V = V_{\max} \cdot (\Delta^g + \Sigma_{\tilde{t}}^r / R_{\max}),$$

with $\Sigma_{\tilde{t}}^r = \frac{1}{2}\sqrt{\frac{C_r \alpha_{\tilde{t}} \Gamma_{\tilde{t}}^r}{\tilde{t}}}$. Then, with high probability,

$$V^{\pi_t}(\boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) \geq V^*(\boldsymbol{s}_t) - \tilde{\epsilon}_V$$

— i.e., the algorithm is $\tilde{\epsilon}_V$-close to the optimal policy — for all but $\tilde{t}$ time steps while guaranteeing safety with probability at least $1 - \Delta^g$.

The proof of Theorem 3 is presented in the supplemental material. The proof is based on the following idea. When the condition in (3.4) is satisfied, the agent will not leave $\mathcal{Y}$, and a near-optimal policy is obtained by optimizing the cumulative reward only in $\mathcal{Y}$ with the optimistically measured reward. Also, as long as the agent is in $\mathcal{Y}$ ($\subseteq \mathcal{X}^-$), safety is guaranteed with high probability. The proof is similar to that for Theorem 2.



```
Run speed = 1.000 x real time    [S]lower, [F]aster
Ren[d]er every frame             On
Switch camera (#cams = 5)        [Tab] (camera ID = 1)
[C]ontact forces                 On
Referenc[e] frames               On
T[r]ansparent                    Off
Display [M]ocap bodies           On
Stop                             [Space]
Advance simulation by one step   [right arrow]
[H]ide Menu
Record [V]ideo (Off)
Cap[t]ure frame
Start [i]pdb
Toggle geomgroup visibility      0-4
```

```
FPS                  11
Solver iterations    1
```

```
Step          61660
timestep      0.00200
n_substeps    1
```

Fig. 3.3 Example screen capture from the GP-Safety-Gym environment.

3.5 Experiment

In this section, we evaluate the performance of SNO-MDP through two experiments. One used a synthetic environment, while the other simulated Mars surface exploration. We also show the effectiveness of our ES^2 and $\mathsf{P\text{-}ES}^2$ algorithms.

3.5.1 Synthetic GP-Safety-Gym Environment

Settings. We constructed a new open-source environment for safe RL simulations named GP-Safety-Gym. This environment was built based on OpenAI Safety-Gym [61]. As shown in Figure 3.3, GP-Safety-Gym represents the reward by a color (yellow: high; green: medium; blue: low), and the safety by height.

We considered a 20×20 square grid in which the reward and safety functions were randomly generated. At every time step, an agent chose an action from *stay*, *up*, *right*, *down*, and *left*. The agent predicted the reward and safety functions by using different kernels on the basis of previous observations. In this simulation, we allowed the agent to observe the reward and safety function values of the current state and neighboring states. The kernel for reward was a radial basis function (RBF) with the length-scales of 2 and prior variance of 1. The kernel for safety was also an RBF with the length-scales of 2 and prior variance of 1. Finally, we set the discount factor to $\gamma = 0.99$, and confidence intervals parameters to $\alpha_t = 3$ and $\beta_t = 2$ for all $t \geq 1$.

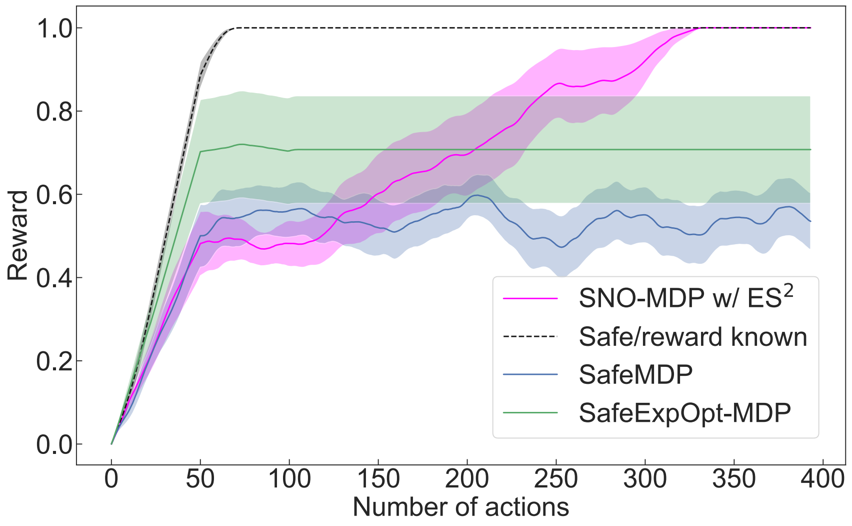


Fig. 3.4 Average reward over the episodes, comparing the performance of SNO-MDP with ES^2 and the baselines.

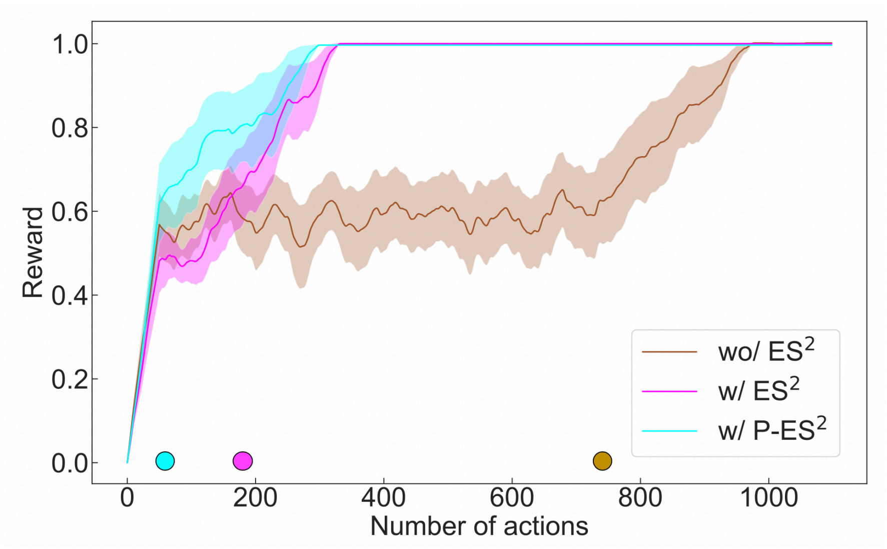


Fig. 3.5 Average reward over the episodes, showing the effects of ES^2 and P-ES^2. The colored circles represent when the transition from safe exploration to reward optimization happens for each method.

Baselines. We empirically compared the performance of our SNO-MDP with

- SAFEMDP (Turchetta et al. [82])
- SAFEEXPOPT-MDP (Wachi et al. [87])
- SAFE/REWARD KNOWN

In SAFEMDP, the agent tries to expand the safe region without considering the reward. In SAFEEXPOPT-MDP, the agent attempts to maximize the cumulative reward while leveraging the difference between the value function in $\mathcal{X}_t^+$ and that in $\mathcal{X}_t^-$. Finally, SAFE/REWARD KNOWN is a non-exploratory case (i.e., oracle agent) in which the safety and reward functions are known a priori.

Metrics. We used the cumulative reward and the number of unsafe actions as comparison metrics to measure the performance of the agent.

Results. Figure 3.4 compares the performance of SNO-MDP and the baselines in terms of the reward. For these results, the average reward was measured over the previous 50 time steps. SNO-MDP achieved the optimal reward after shifting to the stage of reward optimization, which outperforms SAFEMDP and SAFEEXPOPT-MDP in terms of the reward after a sufficiently large number of time steps. The SAFEMDP agent did not aim to maximize the cumulative reward, and the SAFEEXPOPT-MDP agent was sometimes stucked in a local optimum when the expansion of the safe region was insufficient. Figure 3.5 shows the empirical performance of the ES^2 and P-ES^2 algorithms. P-ES^2 achieved faster convergence in terms of the reward than the original ES^2 did. Also, all methods, including the baselines, did *not* take any unsafe actions.

3.5.2 Simulated Mars Surface Exploration

Settings. We then conducted an experiment based on a Mars surface exploration scenario, as in Turchetta et al. [82] and Wachi et al. [87]. In this simulation, we used a publicly available Mars digital elevation model (DEM) that was created from observation data captured by the high-resolution imaging science experiment (HiRISE) camera [48].

We created a 40 $\times$ 30 rectangular grid-world by clipping a region around latitude 30°6' south and longitude 202°2' east, as shown in Figure 3.6. At every time step, the rover took one of five actions: *stay, up, down, left, and right*. We assumed that any state in which the slope angle was greater than 25° were unsafe. The safety function g was defined as the slope angle calculated from the DEM, and the safety threshold was $h = -\tan(25^\circ)$.

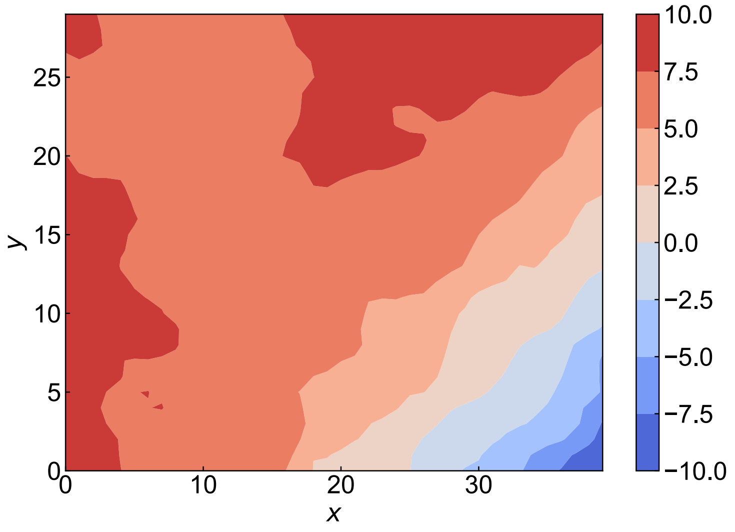


Fig. 3.6 Mars terrain data.

The rover predicted the elevation by using a GP with a Matérn kernel with $\nu = 5/2$. The length-scales were 15 m, and the prior variance over elevation was 100 m^2. We assumed a noise standard deviation of 0.075 m. For the reward, we randomly defined a smooth, continuous reward. To predict the reward function, the rover used a GP with RBF kernel having length-scales of 2 and a prior variance over the reward of 2. We set the confidence levels as $\alpha_t = 3$ and $\beta_t = 2, \forall t \geq 0$, and the discount factor as $\gamma = 0.9$.

Baselines and metrics. We used the same baselines and metrics as in the previous synthetic experiment.

Results. Table 3.1 summarizes the results. The reward was accumulated over the episode, which was normalized with respect to the SAFE/REWARD KNOWN case. Our SNO-MDP with either P-ES^2 or ES^2 outperformed SAFEMDP and SAFEEXPOPT-MDP in terms of the reward. This was expected, because SafeMDP does not aim to maximize the cumulative reward, and SAFEEXPOPT-MDP does not guarantee the near-optimality of the cumulative reward. Also, *no* unsafe action was executed by any of the tested algorithms.

3.6 Conclusion

We have proposed SNO-MDP, a stepwise approach for exploring and optimizing a safety-constrained MDP. Theoretically, we proved a bound of the sample complexity to achieve

Table 3.1 Experimental results with real Mars data.

	Reward	Unsafe actions
SNO-MDP w/ P-ES^2	**0.81**	**0**
SNO-MDP w/ ES^2	**0.78**	**0**
SNO-MDP	**0.49**	**0**
SafeMDP	0.34	0
SafeExpOpt-MDP	0.59	0
Safe/Reward known	1.00	0

ϵ_V-closeness to the optimal policy while guaranteeing safety, with high probability. We also proposed the ES^2 and P-ES^2 algorithms for improving the efficiency in obtaining rewards. We developed an open-source environment, GP-Safety-Gym, to test the effectiveness of SNO-MDP. We also demonstrated the advantages of SNO-MDP using the real Mars terrain data.

3.7 Limitation

In this section, we discuss the limitations of the current algorithm, SNO-MDP. First, our algorithm assumes the regularity and Lipschitz continuity of the safety function as a fundamental scheme for guaranteeing safety. This assumption does not hold in many environments. For example, safety in autonomous driving environments drastically changes depending on the existence of pedestrians or other vehicles. Hence, we need to extend our current algorithms to the one that is applicable to the environment with such steep changes of degree of safety. Second, it is known that the computational cost of a GP is extensive. The computational cost of a GP is $\mathcal{O}(N^3)$, where N is the number of samples. Given that our theoretical result has been achieved after a large number of time steps, it is not desirable to use such an algorithm with the above computational complexity. In order to develop an algorithm that is applicable to large-scale, real-world environments, we need to use less computationally-expensive algorithm. Simple solutions would be to use GPs with small computational cost (e.g., [90, 91]) or (generalized) linear models [52, 66] for characterizing reward and safety functions.

Chapter 4

Policy Testing with Multi-agent Adversarial Reinforcement Learning

If the decision-making algorithm in safety-critical applications does not work properly, the resulting failure may be catastrophic. To prevent such results occurring after deployment, we must determine as many failure cases as possible and then improve the algorithm in the development phase [73]. Autonomous driving and flight algorithms especially must work properly in a multi-agent environment [85, 35], which requires us to craft adversarial situations for the tested algorithm by incorporating the interactions with other agents.

Reinforcement learning (RL) has recently achieved significant results; examples range from robotics manipulation [41] to game playing [50, 68]. However, most of the software in the practical applications are still rule-based because of the explainability or backwards compatibility. This is true with autonomous driving algorithms as well; hence, we need the algorithms to craft adversarial situations for the rule-based algorithm.

As such, it makes sense to train adversarial RL-based agents (i.e., *non-player characters, NPCs*) such that the agent with the tested rule-based algorithm (i.e., *player*) fails. By training NPCs in RL frameworks, we create various adversarial situations without specifying the details of the NPCs' behaviors. We focus on the decision-making aspect rather than image recognition; hence, the failure means collisions in most cases. Figure 4.1 gives a conceptual image of the agents in our research. By training NPCs (red) in an adversarial manner, we aim to obtain the failure cases of the player (blue).

In this problem set, however, we encounter the following four problems. First, pure adversarial training often results in obvious and trivial failure. For example, if multiple NPCs intentionally try to collide with the player, the player will surely fail; however, such failure cases are useless for improving the rule-based algorithm. Second, when the player fails, it is not always clear which NPCs induce the failure. For efficient and stable learning, we

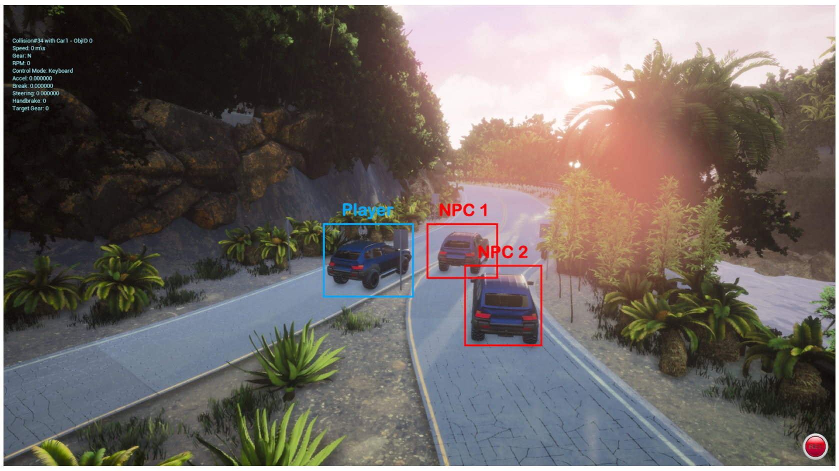


Fig. 4.1 Conceptual image. A blue car is a rule-based player, and two red cars are adversarial RL-based NPCs.

should present the adversarial reward only for the NPCs that contribute to the player's failure. Third, the player does not have a notion of reward, and it is often possible to know only whether or not the player fails. That is, from the perspective of the NPCs, the reward for the player's failure is extremely sparse (e.g., if the player fails, NPCs get the adversarial reward of "1"; otherwise, they get "0").[1] Finally, if the player rarely fails, NPCs are trained using the imbalanced past experience. The imbalanced experience, dominated by the player's success scenarios, prevents NPCs from acquiring a good policy.

Our contributions.

We propose a novel algorithm, FAILMAKER-ADVRL; this approach trains the adversarial RL-based NPCs such that the tested rule-based player fails. In addition, to address the four problems discussed above, we present the following ideas.

Adversarial learning with personal reward. To train NPCs to behave in an adversarial but natural manner, we consider personal reward for NPCs as well as the adversarial reward. This is because we consider that, if it behaves unnaturally, an NPC itself loses the personal reward. When an NPC tries to collide with the player, the personal reward is lost. NPCs should have their own objective (e.g., *safely* arrive at the goal as early as possible), and we consider the loss of the personal reward to ensure natural behavior.

Contributor identification (CI) and adversarial reward allocation (AdvRA). To identify the NPCs that should be provided with an adversarial reward, we propose contributor

[1]We have an option to define a *virtual* reward for the player. However, it is often difficult to precisely define the (virtual) reward.

identification (CI). This algorithm classifies all the NPCs into several categories depending on their degree of contribution by re-running the simulation with the subsets of NPCs. In addition, to handle sparse (episodic) adversarial reward, we propose adversarial reward allocation (AdvRA), which properly allocates the adversarial reward to the NPCs. This algorithm allocates the sparse adversarial reward among each state and action pair that contributes to the player's failures.

Prioritized sampling by replay buffer partition (PS-RBP). To address the problem caused by imbalanced experience, we partition the replay buffer depending on whether or not the player succeeds, and then train the NPCs using the experience that is independently sampled from the two replay buffers.

We demonstrated the effectiveness of FAILMAKER-ADVRL with two experiments using both a simple environment and a 3D autonomous driving simulator.

4.1 Related Work

In this section, we briefly review previous work on MARL with continuous state and action spaces. The relationship among multiple agents can be categorized into cooperative, competitive, and both. Most previous work on MARL addresses cooperative tasks, in which the cumulative reward is maximized as a group [40, 56]. In particular, [23] proposed a method with a centralized critic for a fully cooperative multi-agent task.

Algorithms on MARL applicable with competitive settings have recently been proposed by [58, 44, 42]. [58] consider a two-player zero-sum game in which the protagonist gets a reward r while the adversary gets a reward $-r$. In [44], a centralized critic approach called multi-agent deep deterministic policy gradient (MADDPG) is proposed for mixed cooperative and competitive environments; MADDPG is a similar idea as that in [23]. Finally, Li et al. [42] proposed an algorithm called M3DDPG so that the agent is generalized in terms of the robustness against adversaries' policies alter. In most cases, however, previous studies have aimed to train the protagonist in more robust way while improving the adversary; hence, unlike our method, their objectives are not to obtain failure-scenario of the protagonist (i.e., tested agent).

4.2 Preliminaries

In this section, we briefly present the preliminaries for understanding our method.

Markov games. Partially observable Markov games are a multi-agent extension of POMDPs [43]. A Markov game associated with N agents is defined by a set of states $\mathcal{S}$ meaning all the possible configuration of N agents, a set of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$, and a set of actions $\mathcal{A}_1, \ldots, \mathcal{A}_N$. Agent i decides the action using a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$, by which the next state is produced on the basis of the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathcal{S}$. Each agent i obtains the reward of $r_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ in accordance with the state and agents' action and acquires a new observation $\boldsymbol{o}_i : \mathcal{S} \to \mathcal{O}_i$. The initial states are based on a probabilistic distribution $\rho : \mathcal{S} \to [0, 1]$. Let T be the time horizon. The objective of each agent i is to maximize the expected cumulative reward,

$$R_i = \sum_{t=0}^{T} \gamma^t r_i^t(s, a),$$

where γ is a discount factor.

When the number of agents is more than one, it becomes more difficult for an agent to behave in a good manner. The relationship among multiple agents can be categorized into cooperative, competitive, and both. Most previous work on MARL addresses cooperative tasks, in which the cumulative reward is maximized as a group [40, 56]. In particular, [23] proposed a method with a centralized critic for a fully cooperative multi-agent task. Algorithms on MARL applicable with competitive settings have recently been proposed by [58, 44, 42]. In [44], a centralized critic approach called multi-agent deep deterministic policy gradient (MADDPG) is proposed for mixed cooperative and competitive environments; MADDPG is a similar idea as that in [23].

Multi-Agent Deep Deterministic Policy Gradient (MADDPG). MADDPG is a multi-agent version of the DDPG in which agents learn a centralized critic [44]. MADDPG algorithm is based on the assumption that a policy of an agent can be optimized based on the observations and actions of all agents.

More specifically, consider a game with N agents whose policies are parameterized by $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$, and let $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N\} = \{\boldsymbol{\mu}_{\theta_1}, \ldots, \boldsymbol{\mu}_{\theta_N}\}$ be the set of policies of all agents. Then the gradient of the expected return for agent $i \in [1, N]$ with policy $\boldsymbol{\mu}_i$, $J(\theta_i) = \mathbb{E}[R_i]$ is given by:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{x}, a \sim D}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{1:N})|_{a_i = \boldsymbol{\mu}_i(o_i)}],$$

where $\boldsymbol{a}_{1:N} = \{a_1, \ldots, a_N\}$. For example, $\boldsymbol{x}$ is defined as $\boldsymbol{x} = (o_1, \ldots, o_N)$. In the replay buffer $\mathcal{D}$, the tuples $(\boldsymbol{x}, \boldsymbol{x}', a_{1:N}, r_1, \ldots, r_N)$ (i.e., the recorded experiences of all agents) is contained. The Q-function, $Q_i^{\boldsymbol{\mu}}$ is updated as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\boldsymbol{x},a,r,\boldsymbol{x}'}[(Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{1:N}) - y)^2],$$
$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\boldsymbol{x}', \boldsymbol{a}'_{1:N})|_{a'_l=\boldsymbol{\mu}'_l(o_l)},$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_{\theta'_1}, \ldots, \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with (delayed) parameters θ'_i.

4.3 Problem Statement

We consider a multi-agent problem with a rule-based player and single or multiple NPCs.[2] At every time step, the player chooses an action on the basis of the deterministic rule (i.e. tested algorithm). Our objective is to train the NPCs adversarially to create situations in which the player makes a mistake.

We model this problem as a subspecies of multi-agent Markov games. For a player and N NPCs, this game is defined by a set of states $\mathcal{S}$ meaning all the possible configuration of agents, a set of actions $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_N$, and a set of observations $\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_N$. In the rest of this paper, the subscript "0" represents variables that are for the player. The player chooses an action on the basis of the deterministic rule $\boldsymbol{\mu}_0$, which does not evolve throughout the training of NPCs.[3] Each NPC uses a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$. The next state is produced depending on the state transition $\mathcal{T} : \mathcal{S} \times \mathcal{A}_0 \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathcal{S}$. The player does not obtain a reward from the environment but receives a new observation $\boldsymbol{o}_0 : \mathcal{S} \to \mathcal{O}_0$. Each NPC i obtains a personal reward as the function $\hat{r}_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ in accordance with the state and NPCs' action, and at the same time, receives a new observation $\boldsymbol{o}_i : \mathcal{S} \to \mathcal{O}_i$. The initial states are characterized by a probabilistic distribution $\rho : \mathcal{S} \to [0, 1]$. The ultimate goal of the NPCs is to make the player fail. At the end of each episode, NPCs obtain the binary information on whether or not the player failed.

4.4 FailMaker-AdvRL

Algorithm 2 outlines FAILMAKER-ADVRL. At each iteration, the agents execute the actions on the basis of their policy (Line $2 - 9$). If the player succeeds in the episode, the experience is stored in $\mathcal{D}^+$. If not, the experience is stored in $\mathcal{D}^-$ after identifying the contributing NPCs and allocating the appropriate adversarial reward (Line $10 - 16$). Finally,

[2]Some of the key ideas in this paper can be employed in problem settings where the player is an RL agent.

[3]We assume that $\boldsymbol{\mu}_0$ consists of the numerous number of such deterministic rules as "if the signal is red, then stop."

Algorithm 2 FAILMAKER-ADVRL

```
1: for $e = 1$ to $M$ do
2:     Initialize a random process $\mathcal{N}$ for exploring action
3:     Obtain the initial state $\boldsymbol{x}$
4:     for $t = 1$ to $T$ do
5:         The player executes the action $a_0$ on the basis of the rule $\boldsymbol{\mu}_0$.
6:         For the adversarial NPC $i$, select the action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$
7:         NPCs execute the actions $a = (a_1, \ldots, a_N)$ and observe their personal reward $\hat{r}$
           and next state $\boldsymbol{x}'$
8:         Store $(\boldsymbol{x}, a, \hat{r}, \boldsymbol{x}')$ in temporal replay buffer $\mathcal{D}_{\text{tmp}}$
9:         $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
10:    end for
11:    if the player succeeds then
12:        Samples in $\mathcal{D}_{\text{tmp}}$ are moved to $\mathcal{D}^{+}$
13:        $r_i^t \leftarrow \hat{r}_i^t$
14:    else
15:        After executing CI and AdvRA, the samples in $\mathcal{D}_{\text{tmp}}$ are moved to $\mathcal{D}^{-}$
16:        $r_i^t \leftarrow \hat{r}_i^t + \alpha \cdot \bar{r}_i^t$
17:    end if
18:    for NPC $i = 1$ to $N$ do
19:        Randomly choose $\eta(e)S$ samples from $\mathcal{D}^{+}$ and $(1 - \eta(e))S$ samples from $\mathcal{D}^{-}$ and
           create a random minibatch of $S$ samples with PS-RBP
20:        Update the critic and actor
21:    end for
22:    Update the target network parameters
23: end for
```

the NPCs are adversarially trained using the experiences that are independently sampled from $\mathcal{D}^{+}$ and $\mathcal{D}^{-}$ (Line $17 - 21$). In this section, we first explain three key ideas and then describe the overall algorithm. The detailed pseudocode is given in the supplemental material.

4.4.1 Adversarial Learning with Personal Reward

When we simply train NPCs in an adversarial manner, they will try to make the player fail in whatever way they can. This often results in unnatural situations in which all NPCs try to collide with the player. Considering real applications, it is essentially useless to obtain such unnatural failure cases.

We obtain the player's failure cases by considering a personal reward for NPCs as well as the adversarial reward. For, we consider unnatural situations to be ones in which NPCs themselves lose a large personal reward. Therefore, we train the NPCs while incentivizing them to maximize the cumulative personal and adversarial reward. Let $\hat{r}_i^t$ and $\bar{r}_i^t$ denote the NPC i's personal and adversarial reward. The reward is written as

$$r_i^t = \hat{r}_i^t + \alpha \cdot \bar{r}_i^t, \tag{4.1}$$

where $\alpha \in \mathbb{R}$ is the scaling factor.

When the player fails in an episode, the NPCs receive the adversarial reward. How, though, should we reward each NPC? For efficient and stable training, we should reward the state and action pairs that contribute to the player's failure.

4.4.2 Contributors Identification (CI)

First, to restrict the NPCs that obtain the adversarial reward, we identify the NPCs that contributed to the player's failure. More precisely, for $K \in \mathbb{N}$, we classify NPCs into class k contributors ($k = 1, 2, \ldots, K$). Specifically, class 1 contributors can foil the player alone, and class 2 contributors can foil the player with another class 2 contributor (though they are not class 1 contributors). To identify the class 1 contributors, we re-run the simulation with the player and a single NPC. If the player fails, the NPC is classified as a class 1 contributor. Next, to identify class 2 contributors within the NPCs excluding for the class 1 contributors, we re-run the simulation with the player and two NPCs while seeding the randomness of the simulation. The above steps are continued until we identify the class K contributors. In the rest of the paper, we denote $\mathcal{C}_k$ as the set of class k contributors and $\mathcal{C}$ as the set of contributors; that is,

$$\mathcal{C} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_K.$$

When identifying the class k contributors, the number of simulations is ${}_N C_k$; hence, K should be small for a large N. Practically, since traffic accidents are caused by the interaction among a small number of cars, setting small K does not usually affect the practicality.

4.4.3 Adversarial reward allocation (AdvRA).

After identifying the contributors, we assign the adversarial reward to each state and action pair. Let g denote the measure of the contribution to the player's failure, which we will call the *contribution function*. Suppose NPC i^* is the biggest contributor to the player's failure. The biggest contributor is identified by:

$$i^* = \underset{i \in \mathcal{C}_{k_{\min}}}{\arg\max} \left(\max_t g_t^i(\boldsymbol{x}, a_i) \right),$$

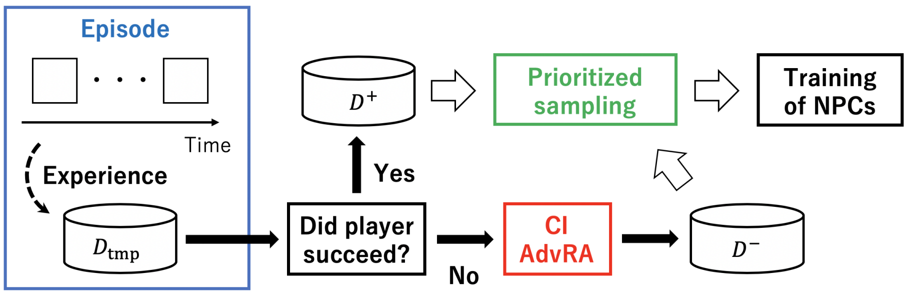


Fig. 4.2 Conceptual image of PS-RBP.

where $k_{\text{min}} = \min\{k \in [1, K] \mid \mathcal{C}_k \neq \emptyset\}$. Here, we allocate the adversarial reward to each state and action pair of the NPCs as follows. First, the adversarial reward of "1" is allocated to each state and action pair of NPC i^* depending on the contribution function g:

$$\bar{r}^t_{i^*}(\boldsymbol{x}, a_{i^*}) = g^t_{i^*}(\boldsymbol{x}, a_{i^*}) / \sum_t g^t_{i^*}(\boldsymbol{x}, a_{i^*}).$$

The adversarial reward is then allocated to the all contributors, $\mathcal{C}$ in accordance with their contributions. For NPC $i \in \mathcal{C}_k$, we set

$$\bar{r}^t_i(\boldsymbol{x}, a_i) = w(k) \cdot \bar{r}^t_{i^*}(\boldsymbol{x}, a_{i^*}) \cdot g^t_i(\boldsymbol{x}, a_i) / g^t_{i^*}(\boldsymbol{x}, a_{i^*}), \tag{4.2}$$

where $w(k) \in [0, 1]$ is the monotone non-increasing function with regard to k, which represents the weight between the classes of the contributors. Empirically, to prevent NPCs from obstructing other NPCs, w should not be too small. In our simulation, the contribution function g^t_i is simply defined using the distance between the player and NPC i; that is,

$$g^t_i = \exp(-\beta \cdot \|\boldsymbol{s}^t_0 - \boldsymbol{s}^t_i\|), \tag{4.3}$$

where β is a positive scalar, and $\boldsymbol{s}^t_0$ and $\boldsymbol{s}^t_i$ are the positions of the player and NPC i, respectively. Intuitively, this contribution function is based on the consideration that the action inducing the player's failure should be taken when NPC i is close to the player. An alternative would be to define the contributor function using commonly used safety metrics such as headway and time-to-collision [86].

4.4.4 Prioritized Sampling by Replay Buffer Partition (PS-RBP)

In the case that the player rarely fails, most of the experiences in the replay buffer are ones in which the player succeeds. As a result, we are required to train adversarial NPCs with imbalanced experiences. To address this problem, we propose PS-RBP. We partition the replay buffer into two parts to separate the experience according to whether or not the player succeeds. During an episode, the experience is temporally stored in $\mathcal{D}_{\text{tmp}}$. After finishing the episode, the experiences in $\mathcal{D}_{\text{tmp}}$ are transferred to $\mathcal{D}^{+}$ if the player succeeds and to $\mathcal{D}^{-}$ if the player fails. A conceptual image is shown in Figure 4.2.

Let e denote the episode number. In training the NPCs, we employ $\eta(e) \cdot S$ samples from $\mathcal{D}^{+}$ and $(1 - \eta(e)) \cdot S$ samples from $\mathcal{D}^{-}$, where $\eta(e) \in [0, 1]$ is the coefficient between the number of samples from $\mathcal{D}^{+}$ and $\mathcal{D}^{-}$. S is the number of samples employed in the training of the neural network.

4.4.5 Overview of FAILMAKER-ADVRL

An overall structure of FAILMAKER-ADVRL is shown in Figure 4.3. As a framework for training multiple agents, we employ the MADDPG algorithm proposed in [44]. The MADDPG algorithm is a type of multi-agent PG algorithm that works well in both cooperative and competitive settings. MADDPG is a decentralized-actor-and-centralized-critic approach, which allows the critic to know the observations and policies of all agents. In this paper, we also allow the critic to know the observations and policies of the player and all NPCs. We consider applying FAILMAKER-ADVRL in the development of the player's rule; hence, this assumption is not restrictive.

Suppose that the policies of N NPCs are parameterized by $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_N\}$. Also, let $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N\} = \{\boldsymbol{\mu}_{\theta_1}, \dots, \boldsymbol{\mu}_{\theta_N}\}$ be the set of all agent policies. The gradient of the expected cumulative reward for NPC i, $J(\theta_i) = \mathbb{E}[R_i]$ is written as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{x}, a \sim \mathcal{D}^{\pm}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{0:N})|_{a_0 = \boldsymbol{\mu}_0, a_i = \boldsymbol{\mu}_i}],$$

where $\boldsymbol{a}_{0:N} = \{a_0, \dots, a_N\}$. $\mathcal{D}^{\pm}$ represents the replay buffer partitioned into $\mathcal{D}^{+}$ and $\mathcal{D}^{-}$; that is, the experience is sampled from the two replay buffers using PS-RBP. Note that the experiences in $\mathcal{D}^{-}$ contains the reward after executing CI and AdvRA.

Using the reward function in (4.1) characterized by the personal and adversarial reward, the action-value function $Q_i^{\boldsymbol{\mu}}$ is updated for all NPC i as follows:

$$\begin{aligned}\mathcal{L}(\theta_i) &= \mathbb{E}_{\boldsymbol{x}, a, r, \boldsymbol{x}'}[(Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{0:N}) - y)^2], \\ y &= r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\boldsymbol{x}', \boldsymbol{a}'_{0:N})|_{a'_0 = \boldsymbol{\mu}'_0(o_0), a'_l = \boldsymbol{\mu}'_l(o_l)},\end{aligned}$$

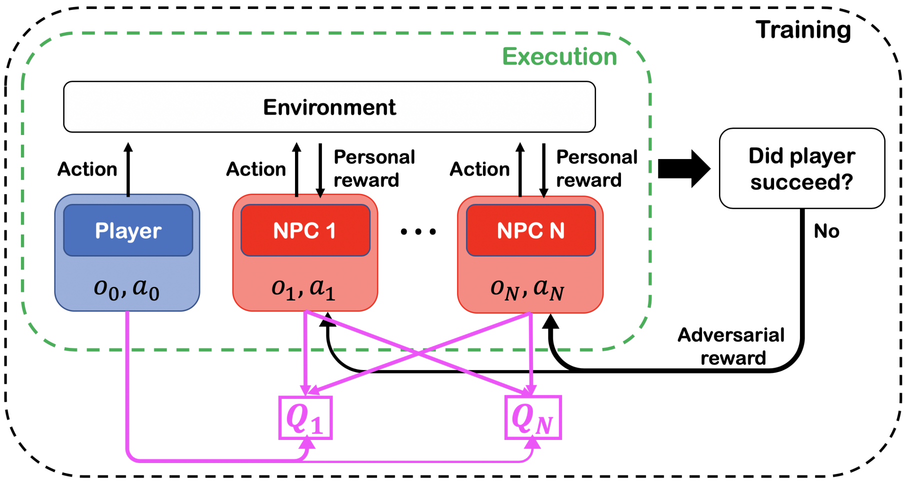


Fig. 4.3 Problem with a player and N NPCs. NPCs can get the adversarial reward when the player fails.

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_0, \boldsymbol{\mu}_{\theta'_1}, \ldots, \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with delayed parameters θ'_i. Practically, we employ a soft update as in

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i,$$

where τ is the update rate.

Note that our proposed AdvRA and CI are varieties of credit assignment methods [18, 54]. Most existing methods allocate the proper reward to each agent by utilizing difference reward [92] under the assumption that all agents are trained by RL. Also, prioritized experience-sampling techniques were previously proposed in [62] and [29]; these techniques enable efficient learning by replaying important experience frequently.

4.5 Diverse FailMaker-AdvRL

FAILMAKER-ADVRLis a method for finding failure-scenarios, but there is one critical issue; that is, failure-scenarios created by FAILMAKER-ADVRLare not diverse. Let us consider what kinds of characteristics *valuable failure-scenarios* are useful for improving or debugging your tested algorithm. We consider the following instances of such characteristics.

- Diversity: Diverse failure-scenarios should be obtained.
- Possibility: Unrealistic failure-scenarios are useless.
- Severity: More severe failure-scenarios should be found.

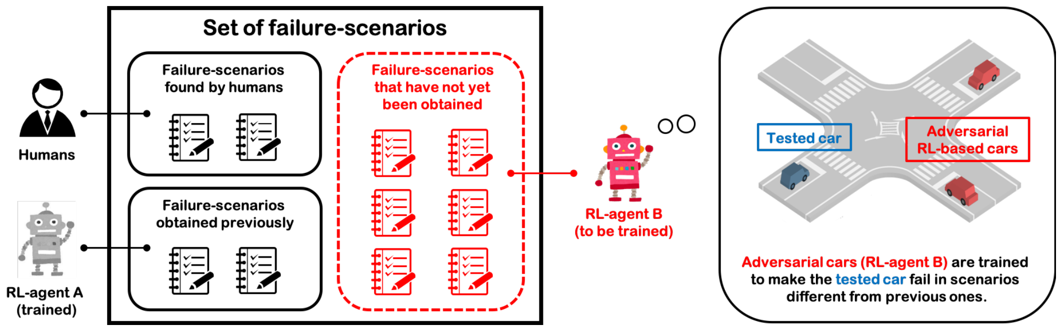

Set of failure-scenarios
Failure-scenarios found by humans
Failure-scenarios obtained previously
Failure-scenarios that have not yet been obtained
Humans
RL-agent A (trained)
RL-agent B (to be trained)
Tested car
Adversarial RL-based cars
Adversarial cars (RL-agent B) are trained to make the tested car fail in scenarios different from previous ones.

Fig. 4.4 Conceptual image of our problem. Suppose failure-scenarios have already been obtained by humans and trained agents (i.e., RL-agent A). Our ultimate objective is to find diverse failure-scenarios. As such, we train other agents (i.e., RL-agent B) to make the tested car fail in scenarios different from the ones that were previously obtained.

- Commonality: failure-scenarios that help improve or debug the fundamental part of the tested algorithm or system are preferable.

When testing software or algorithms for a safety-critical application, we should obtain the failure-scenarios that are rich in quantity and quality.

In this paper, we focus on the problem of *diversity*. For example, are one hundred similar failure-scenarios really useful? Similar failure-scenarios are essentially worth a single failure-scenario. To enhance the reliability and quality of the tested algorithm, it is significant to find diverse failure-scenarios and improve the algorithm so that it does not fail in situations similar to the ones previously obtained.

As such, we consider applying multi-agent reinforcement learning (MARL) in order to obtain scenarios different from the ones that were previously obtained. More concretely, we train adversarial RL-based agents to make the tested agent fail in a scenario different from the ones obtained by humans or previously trained RL-based agents. One advantage of using RL is that we do not have to manually tune the various parameters of the agents. Conceptual image of this work is shown in Figure 4.4.

DIV-FAILMAKER-ADVRL is outlined in Algorithm 3 and illustrated in Figure 4.5. First of all, we aim to get the diverse failure-scenarios by updating the *generation* of the NPCs; that is, we train m-th generation of NPCs to make the player fail in scenarios different from those obtained by the first to $(m-1)$-th generation of NPCs, where $m \in \mathbb{Z}$ denotes the generation. When updating the generation of the NPCs, we identify the key variables for diversity (KVfD), and then train the m-th generation of NPCs to create failure-scenarios with different KVfDs from previous ones.

At each iteration, the agents (the player and the NPCs) execute actions based on their policy (Line 3−8). After calculating the KVfDs, the experiences $(\boldsymbol{x}, a, r, \boldsymbol{x}', b)$ are stored in

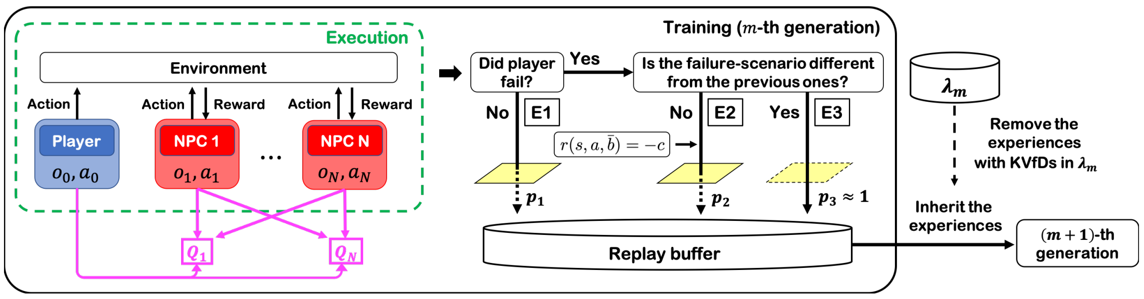


Fig. 4.5 Illustration of DIV-FAILMAKER-ADVRL. After each episode, the experiences are categorized and stored in the replay buffer, $\mathcal{D}$ with the pre-defined probability based on the category. For the experiences categorized with E2, the reward is set to non-positive value, $-c$.

the temporal replay buffer $\mathcal{D}_{\text{tmp}}$ (Line 9). To prevent the replay buffer $\mathcal{D}$ from being filled with experiences in which the player does not fail or similar failure-scenarios, we dump the less useful experiences with a probability depending on the category (Line 12−22). Also, if the KVfD of the currently obtained failure-scenario is similar to the one in the previously obtained failure-scenarios, the reward is set to a negative value or zero (Line 17). Finally, the NPCs (i.e., the critic and actor) are trained using the experiences that are sampled from the replay buffer (Line 23−29). After the training of the NPCs is terminated, the set of the registered KVfDs (Reg-KVfDs) is updated (Line 31), then the experiences in the replay buffer are transferred to the next generation (Line 32).

In this section, we explain several key ideas, then describe the overall algorithm.

4.5.1 Key Variables for Diversity (KVfD)

Our objective is to find the diverse failure-scenarios; that is, we need to make the following ϕ diverse:

$$\boldsymbol{\psi} = (\boldsymbol{x}^{0:T}, \boldsymbol{a}_0^{0:T}, \ldots, \boldsymbol{a}_N^{0:T})$$

How then, can we make $\boldsymbol{\psi}$ diverse?

In a competitive multi-agent reinforcement learning setting, the next states of agents (i.e., $\boldsymbol{x}'$) are determined by the dynamics of each agent and the interactions between agents. Because the failure-scenarios are created by a combination of each agent's actions and their interactions, making the failure-scenarios diverse directly is particularly difficult.

We gained insight into the characteristic of certain variables tending to be similar. In the autonomous driving testing example, the position of the player's failure tended to be especially similar when there were unsafe spots in the environment or the player had weak points. Therefore, we introduce a new variable b and denote the set of b as B. Here, the probabilistic distribution of failure-scenarios $P(\Psi)$ satisfies

$$P(\Psi) = \int_B P(\Psi \mid B) P(B).$$

In this paper, b represents the variable that tends to be similar among similar scenarios and we call b the *key variables for diversity (KVfD)*. For example, in our experiment, b represents the player's positions of the player at the moment of its failures (i.e., collisions). Intuitively, we aim to create diverse failure-scenarios by making $P(B)$ as uniform as possible. As for $P(\Psi \mid B)$, we achieved diversity by randomly choosing the initial state and/or changing the random noise for action exploration.

Also, we define Reg-KVfDs as the set of KVfDs *registered* as the ones that are frequently obtained by previously trained NPCs.

4.5.2 Reward Function Characterized by KVfD

To make the KVfD more diverse, we define the reward function as $r(\boldsymbol{x}, a, b)$ using the KVfD as well as the state-action pair. Supposing we now train m-th generation NPCs and let λ_{m-1} denote the set of the Reg-KVfD whose failure-scenarios are often created by $(m-1)$-th generation NPCs, we also denote Λ_{m-1} as the set of Reg-KVfD already obtained by the (previously trained) first to $(m-1)$-th generation NPCs; that is, $\Lambda_{m-1} = \lambda_1 \cup \ldots \cup \lambda_{m-1}$. We train m-th generation NPCs in failure-scenarios different from those obtained by previous generation NPCs. For diverse failure-scenarios, we set the reward as a non-positive value if the KVfD of the failure-scenario is similar with the one in Λ_{m-1} as follows:

$$r(\boldsymbol{x}, a, \bar{b}) = -c \quad \text{if} \quad \min_{b \in \Lambda_{m-1}} \|b - \bar{b}\| \leq h, \tag{4.4}$$

where $\bar{b}$ is the currently obtained KVfD, $c \in \mathbb{R}$ is a non-negative scalar, and $h \in \mathbb{R}$ is a threshold. We explain how to determine λ and Λ below.

4.5.3 Determining Reg-KVfD

When the training of m-th generation of NPCs is terminated (i.e., converged), the set of Reg-KVfDs, Λ, is updated with $\Lambda_m = \Lambda_{m-1} \cup \lambda_m$. Here, we present two methods of identifying the Reg-KVfD at the m-th generation NPCs (i.e., λ_m). The first method is simply performing a Monte-Carlo simulation with the trained NPCs; the second is using the scenarios obtained in the last several percent of the episodes. Both methods are count-based; that is, we create a histogram with regard to b, then identify them as Reg-KVfDs if the number of b (or the ratio of b regarding the number of test episodes) is more than a threshold.

4.5.4 Experience Storing with Filtration (ESwF)

The experience is temporally stored in $\mathcal{D}_{\text{tmp}}$ during an episode, then transferred to the replay buffer $\mathcal{D}$ afterwards. If we simply transfer all the experiences from $\mathcal{D}_{\text{tmp}}$ to $\mathcal{D}$, most of the experiences in $\mathcal{D}$ might be ones in which the player succeeds or the player fails in the scenarios similar to the ones that were previously obtained. To find the diverse failure-scenarios, we should store and sample the experiences in which the player fails in scenarios different from the previously obtained ones.

As such, we propose the Experience Storing with Filtration (ESwF) algorithm. This algorithm sorts all the experiences into three categories (E1, E2, and E3). Category E1 is when the player succeeds, E2 is when the player fails in a scenario similar to a previously obtained one, and E3 is when the player fails in a new scenario.

Then, we transfer the experiences from $\mathcal{D}_{\text{tmp}}$ to $\mathcal{D}$ with a probability $p_{\{1,2,3\}} \in [0, 1]$, where $p_{\{1,2,3\}}$ is a priori allocated value for each experience category (i.e., E1, E2, and E3). Note that p_3 should be $p_3 \approx 1$ because the experiences categorized as E3 are particularly useful for training and not frequently obtained. We dump the experiences that are not transferred to $\mathcal{D}$ are dumped.

4.5.5 Inheriting the Experience

In many cases, E3 experiences are not frequently obtained, so we need to collect sample experiences by running a number of episodes in the early stage of the training for each generation. Therefore, after calculating the Reg-KVfDs for the m-th generation of NPCs, we take the experiences over to the $(m + 1)$-th generation. However, to prevent later generations from making the player fail in similar ways to m-th generation, the experiences with the λ_m are removed without being inherited. By reusing the experiences, we can efficiently train later generations.

4.5.6 Overview of DIV-FAILMAKER-ADVRL

The training structure of the NPCs is on the left in Figure 4.3. We adopt the MADDPG algorithm [44] as a framework for training multiple agents.

As with the MADDPG algorithm, DIV-FAILMAKER-ADVRL is a decentralized-actor-and-centralized-critic approach, which means we allow the NPCs to get access to the observations and actions of the player and all NPCs (i.e., $\{o_0, a_0, o_1, a_1, \ldots, o_N, a_N\}$). We consider applying DIV-FAILMAKER-ADVRL when developing the player's algorithm that are implemented in a fully simulated virtual environment; hence, this assumption is not restrictive.[4]

Suppose that the policies of N NPCs are parameterized by $\boldsymbol{\theta}_{1:N} = \{\theta_1, \ldots, \theta_N\}$. Also, let

$$\boldsymbol{\mu}_{1:N} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N\} = \{\boldsymbol{\mu}_{\theta_1}, \ldots, \boldsymbol{\mu}_{\theta_N}\}$$

be the set of all agent policies. The gradient of the expected cumulative reward for NPC i is calculated as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{x}, a \sim \mathcal{D}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{0:N})|_{a_0 = \boldsymbol{\mu}_0, a_i = \boldsymbol{\mu}_i}],$$

where $\boldsymbol{a}_{0:N} = \{a_0, \ldots, a_N\}$, and the replay buffer $\mathcal{D}$ contains the experiences stored with the probability $p_{\{1,2,3\}}$ using the ESwF. The centralized action-value function $Q_i^{\boldsymbol{\mu}}$ is updated for all NPC i using the reward function defined in (4.4) as follows:

$$\begin{aligned}\mathcal{L}(\theta_i) &= \mathbb{E}_{\boldsymbol{x}, a, r, \boldsymbol{x}'}[(Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{a}_{0:N}) - y)^2], \\ y &= r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\boldsymbol{x}', \boldsymbol{a}'_{0:N})|_{a'_0 = \boldsymbol{\mu}'_0(o_0), a'_l = \boldsymbol{\mu}'_l(o_l)},\end{aligned}$$

where $\boldsymbol{\mu}'_{0:N} = \{\boldsymbol{\mu}_0, \boldsymbol{\mu}_{\theta'_1}, \ldots, \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with delayed parameters θ'_i. Specifically, a soft update is employed as in

$$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i,$$

where $\tau \in \mathbb{R}$ is the update rate.

Note that NPC training is conducted from scratch for each generation despite inherited experiences. Also, the reward function is consistent in each generation.

4.6 Experiments for FAILMAKER-ADVRL

We present empirical results from two experiments. The first is in a multi-agent particle setting, and the second is in an autonomous driving setting.

Table 4.1 Percentage of failures of player and NPC(s). FAILMAKER-ADVRL outperforms other baselines in terms of its ability to foil the player while constraining the percentage of failures of NPCs.

	Simple environment					
	$N = 1$		$N = 3$			
	Player	NPC	Player	NPC 1	NPC 2	NPC 3
Good Agent	2.4	1.3	5.2	1.6	1.1	1.3
Attacker	98.9	98.0	100.0	98.0	99.5	99.2
P-Adv	8.6	1.2	12.9	1.5	1.6	1.1
P-Adv_AdvRA	60.8	1.2	74.3	1.7	1.6	1.6
P-Adv_AdvRA_CI	79.6	1.0	82.9	1.5	1.4	1.9
P-Adv_PS-RBP	11.4	0.9	23.0	1.4	1.5	0.9
FAILMAKER-ADVRL	**95.6**	**1.0**	**99.2**	**1.4**	**1.2**	**1.1**

4.6.1 Simple Multi-agent Particle Environment

We first tested FAILMAKER-ADVRL using simple multi-agent environments [53]. In this simulation, we consider a player and N NPCs with $N = 1$ and 3. Put simply, the player attempts to approach the goal on the shortest path. The player's goal is set as $(0, 0.7)$. However, when the player gets close to any NPCs, the player acts to maintain its distance from the NPC. In contrast, NPCs get the adversarial reward at the end of the episode if the player collides with another agent or a wall (i.e. a boundary) in at least one time step. At the same time, NPCs receive the personal reward by staying close to the goals. NPCs also try to avoid losing their personal reward; we do this to maintain the NPCs' natural behavior. More specifically, the personal reward decreases when arriving at the goal late or colliding with another agent or wall. For the $N = 1$ case, the goal of the NPC is set as $(0.5, 0)$. For the $N = 3$ case, the goals are set as $(\pm 0.5, 0)$ and $(0, -0.5)$.

We implemented FAILMAKER-ADVRL. Our policies are modeled by a two-layer ReLU neural networks with 64 units per layer. In our experiment, The optimizer was the Adam optimizer with a learning rate of 0.01. The sizes of $\mathcal{D}^+$ and $\mathcal{D}^-$ are both 1×10^6, and the batch size is set as 1024 episodes. In addition, the discount factor, γ is set to 0.95. For $N = 1$ and 3, K is set to $K = 1$ and 2, respectively. The contribution function g is defined as in (4.3) with $\beta = 2.0$. The weight w in (4.2) is set as $w(k) = 0.9^{k-1}$. In our simulation, through trial and error, $\eta(e)$ in PS-RBP is set as $\eta = 0.5$ for $e < 2500$ and $\eta = 0.25$ otherwise.

[4]We suppose that DIV-FAILMAKER-ADVRL is used in model-in-the-loop (MIL) or software-in-the-loop (SIL) simulators rather than in hardware-in-the-loop (HIL) simulators or field tests.

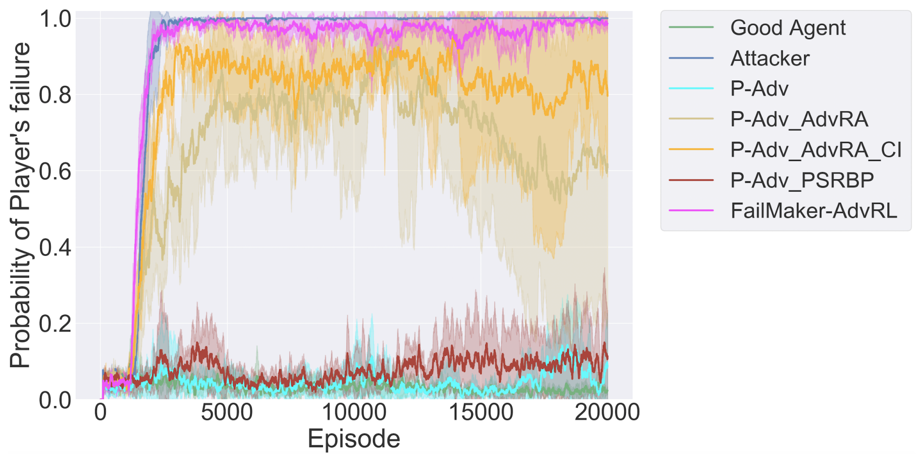


Fig. 4.6 Comparing performance in simple multi-agent particle environment with one NPC. The player's failure rate is measured over the previous 100 episodes.

Baselines. We compared FAILMAKER-ADVRL with the following six baselines.

- **Good Agent**: Good NPCs that maximize their cumulative personal reward (i.e. only $\hat{r}$ is considered).
- **Attacker**: Pure adversaries that maximize the cumulative adversarial reward (i.e. only $\bar{r}$ is considered).
- **P-Adv**: Adversaries with the personal reward.
- **P-Adv_AdvRA**: Adversaries with the personal reward using AdvRA (without PS-RBP and CI).
- **P-Adv_AdvRA_CI**: Adversaries with the personal reward using AdvRA and CI (without PS-RBP).
- **P-Adv_PS-RBP**: Adversaries with the personal reward using PS-RBP (without AdvRA and CI).

We used the same parameters as in FAILMAKER-ADVRL for the above baselines. For the baselines without PS-RBP, the size of the replay buffer was 1×10^6.

Metrics. We used the following as the metrics: 1) the number of the player's failures, and 2) the number of NPCs' failures. The first metric measures the quality of being adversarial, and the second measures the naturalness (i.e. usefulness) of the adversarial situations.

Results. The left half in Table 4.1 compares the test performance of FAILMAKER-ADVRL and that of the baselines. Each value indicates the percentage of failure in 1000 episodes. For both $N = 1$ and 3, FAILMAKER-ADVRL outperforms the other baselines except for Attacker in terms of its ability to make the player fail. Attacker has a larger percentage for the player's failure, but this is because Attacker intentionally tries to collide with the player. Figure 4.6 represents the percentage of the player's failure over the number of episodes. Observe that FAILMAKER-ADVRL and Adversary achieve stable and efficient learning. Also note that P-Adv_AdvRA and P-Adv_AdvRA_CI execute relatively late convergence compared with FAILMAKER-ADVRL.

4.6.2 Autonomous Driving

We then applied FAILMAKER-ADVRL to the multi-agent autonomous driving problem using Microsoft AirSim [67]. We consider a rule-based player and N adversarial RL-based NPCs. We tested for $N = 1$ and 2. In this simulation, the objective of the NPCs is to make the player cause an accident or arrive at the goal destination very late.

In this simulation, we assumed that the player and NPCs have reference trajectories, which are denoted as $\Xi_i = \{\boldsymbol{\xi}_i^1, \ldots, \boldsymbol{\xi}_i^T\}$, where $\boldsymbol{\xi}_i^t$ is the reference position and orientation (i.e. yaw angle) at time t for agent i. In our experiment, the reference trajectory is obtained through manual human demonstration. We used the coastline environment in the AirSim simulator, so all the agents drive on a two-lane, one-way road. For simplicity, we assumed that 1) the reference trajectories of all agents are identical (i.e., $\Xi_0 = \ldots = \Xi_N$) and that 2) all agents can observe the true state at every time step.

The player aims to follow the reference trajectory, Ξ_0 using PD feedback control. Depending on the difference between the actual state $\boldsymbol{\zeta}_0^t$ and the reference state $\boldsymbol{\xi}_0^t$, the player chooses its own action. Therefore, the player follows the reference trajectory by feed-backing $\boldsymbol{\zeta}_0^t - \boldsymbol{\xi}_0^t$. The player also avoids colliding with NPCs. When the player gets closer to NPCs than a threshold, the player acts to keep its distance from the closest NPC. NPCs act to optimize their personal reward (i.e., $\hat{r}$) and adversarial reward (i.e., $\bar{r}$). The NPCs' personal reward is defined using 1) the distance with the reference trajectory and 2) the velocity. In other words, NPCs are trained to keep to the center of the read and arrive at the destination in a short time. Also, NPCs are trained to behave in an adversarial way on the basis of the adversarial reward.

The FAILMAKER-ADVRL algorithm is implemented as follows. Our policies are modeled by a three-layer ReLU neural networks with 256 units per layer. We used the Adam with a learning rate of 0.01. The sizes of $\mathcal{D}^+$ and $\mathcal{D}^-$ are both 1×10^6, and the batch size is set as

Table 4.2 Percentage of failures of player and NPC(s). FAILMAKER-ADVRL outperforms other baselines in terms of its ability to foil the player while constraining the percentage of failures of NPCs.

	Autonomous driving				
	$N = 1$		$N = 2$		
	Player	NPC	Player	NPC 1	NPC 2
Good Agent	3.4	2.9	5.2	3.0	2.0
Attacker	98.0	98.0	100.0	98.1	98.2
P-Adv	5.4	1.9	6.9	2.5	2.9
P-Adv_AdvRA	42.3	1.8	56.7	1.7	2.1
P-Adv_AdvRA_CI	65.6	2.0	69.2	3.1	3.3
P-Adv_PS-RBP	8.1	2.6	8.5	3.6	3.1
FAILMAKER-ADVRL	**78.1**	**2.8**	**84.5**	**3.2**	**3.4**

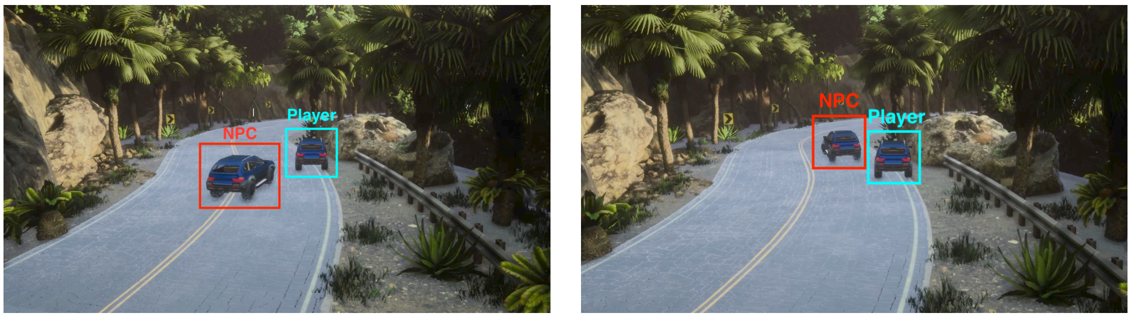


Fig. 4.7 Example of failure of player (blue) induced by adversarial NPC (red).

1024 episodes. Also, for the discount factor, γ is set to be 0.95. We used the same baselines and metrics as in Section 4.6.1 to evaluate FAILMAKER-ADVRL.

Results. The right half in Table 4.2 compares FAILMAKER-ADVRL and the baselines. Each value indicates the failure rate of the player and NPCs in 1000 episodes. FAILMAKER-ADVRL outperforms the other baselines except for Attacker in terms of its ability to make the player fail. Figure 4.7 shows an example of the simulation results using the 3D autonomous driving simulator. An adversarial NPC successfully induces the player's collision with a rock. Note that the NPC does not collide with the player or any obstacles. A sample video is in the supplemental material.

Table 4.3 Covariance matrices for Pure-CompetMARL and DIV-FAILMAKER-ADVRL.

	Covariance matrix
Pure-CompetMARL	$\begin{bmatrix} 7.5 \times 10^{-4} & 1.2 \times 10^{-4} \\ 1.2 \times 10^{-4} & 1.8 \times 10^{-4} \end{bmatrix}$
Proposed ($m \leq 2$)	$\begin{bmatrix} 2.9 \times 10^{-2} & -3.9 \times 10^{-3} \\ -3.9 \times 10^{-3} & 1.6 \times 10^{-3} \end{bmatrix}$
Proposed ($m \leq 5$)	$\begin{bmatrix} 1.7 \times 10^{-2} & 1.4 \times 10^{-2} \\ 1.4 \times 10^{-2} & 4.8 \times 10^{-2} \end{bmatrix}$

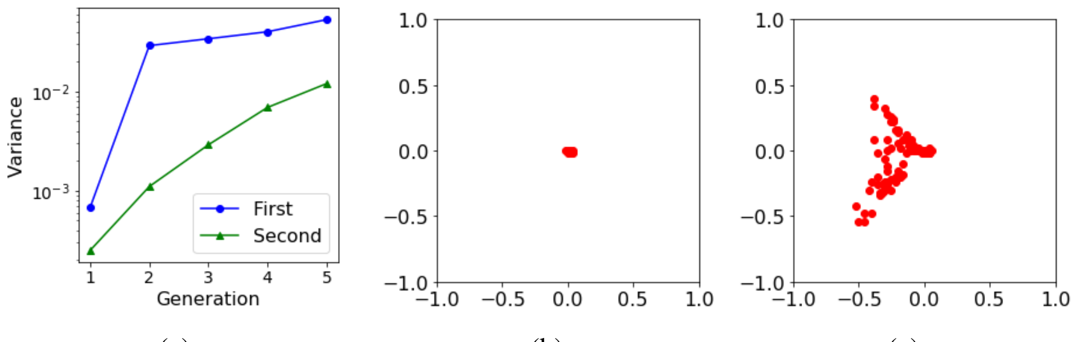
4.7 Experiments for DIV-FAILMAKER-ADVRL

We tested our DIV-FAILMAKER-ADVRL in a 2D multi-agent particle environment [53].

Settings. The problem we used consisted of a player and an NPC. At every time step, the rule-based deterministic policy, $\boldsymbol{\mu}_0$ made the player head for its own goal. We trained an NPC to make the player fail. Collisions between the player and another agent or a boundary (e.g., a wall) were regarded as failures. We set the player's goal as $(0, 0)$. We set the x-coordinate to -0.8 for the initial positions of the player and randomly defined the y-coordinate within the range of $[-1, 1]$ We randomly defined the initial position of the NPC within the range of $[-0.5, 0.5]$. We made the relationship between the player and the NPC imbalanced; that is, the NPC could know the observations and actions of all agents. Also, to make it easier for the NPC to make the player fail, we set the NPC's maximum velocity higher than player's.

Implementation. In this experiment, we implemented the DIV-FAILMAKER-ADVRL algorithm as follows. The policy of the NPC was parameterized by a three-layer ReLU multi-layer perceptron (MLP) with 64 units per layer. In our experiment, The optimizer was the Adam with a learning rate of 0.01 and $\tau = 0.01$ for the soft update of the target network. We made the size of the replay buffer 1×10^6, and the batch size 1024. We set the filtration ratios for the E1 experience category as $p_1 = 0.1$, E2 as $p_2 = 1.0$, and E3 as $p_3 = 1.0$. We set the discount factor to $\gamma = 0.95$, the maximum episode length to $e_{\max} = 30$. The cost and the threshold in (4.4) are set to $c = 1$ and $h = 0.1$, respectively. We set the number of episodes for to 3.5×10^4 per generation and the maximum generation to $m_{\max} = 5$.

Reg-KVfDs were identified as follows. After terminating the training of the NPC, we ran the simulation 5,000 times, then created a 2D histogram based on b with 100 bins for both x and y axes. If the proportion of b for a bin to the total number of failure cases exceeded 0.5%, we identified them as Reg-KVfDs.



(a) (b) (c)

Fig. 4.8 (a) Relationship between generation and variance of the matrix transformed using PCA. (b, c) Plot of the Reg-KVfDs (i.e., collision location of the player). Each figure is for (b) Pure-CompetMARL and (c) DIV-FAILMAKER-ADVRL ($m \leq 5$).

Baselines. We compared our DIV-FAILMAKER-ADVRL with the pure adversarial RL without diversity (i.e., **Pure-CompetMARL**). More specifically, in this baseline, the NPC was given the reward regardless of whether or not the player failed in similar scenarios (with similar KVfD). We expected Pure-CompetMARL to create similar failure-scenarios of the player because the NPC tried to make the player fail in the easiest way for them. We also set the number of episodes for Pure-CompetMARL to 3.5×10^4, which is identical with that for each generation of our method. Also, other parameters were identical with DIV-FAILMAKER-ADVRL.

Metrics. We measured the diversity of the failure-scenarios with two metrics. The first metric is simply the covariance matrix of the KVfDs, and the second metric is the diagonalized matrix obtained using principle component analysis (PCA).

Results. Figure 4.8(a) represents the relationship between the variance and the generation of the NPC. The blue line is on the variance for the first principle component, and the green line is for the second component, obtained using PCA. The first and second principle components both monotonically increase with the generation. Also, Figure 4.8(b-c) represents the plot of Reg-KVfDs for Pure-CompetMARL and DIV-FAILMAKER-ADVRL. Table 4.3 shows the original covariance matrices regarding $\boldsymbol{\Lambda}$ for Pure-CompetMARL and DIV-FAILMAKER-ADVRL.

In Pure-CompetMARL, the (trained) NPC tries to make the player fail at the origin (i.e., the destination of the player) because just waiting is the easiest way for the NPC. However, DIV-FAILMAKER-ADVRL succeeds in making the player fail at different locations; that is, the NPC diversifies the KVfDs as the generation increases.

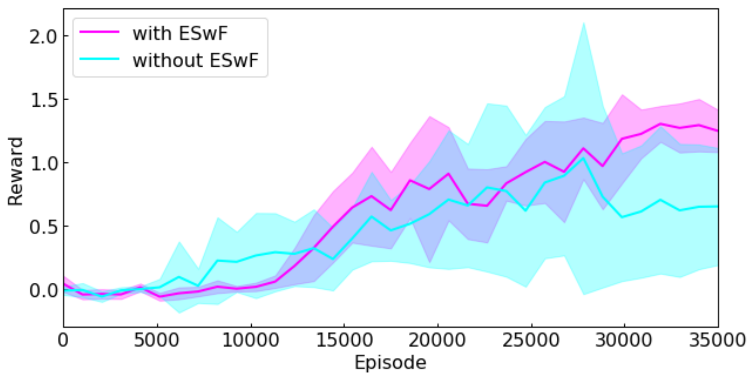


Fig. 4.9 Effect of ESwF (magenta: with ESwF; cyan: without ESwF). The cumulative reward is measured over the 1,000 previous episodes.

Effects of ESwF. Figure 4.9 compares the cumulative reward of DIV-FAILMAKER-ADVRL with ESwF and without ESwF and shows that we achieve a higher expected cumulative return with a small standard deviation with ESwF. When we did not use ESwF, category E1 (i.e., the experiences in which the player succeeded) dominated the replay buffer, $\mathcal{D}$. By employing ESwF, the balanced experiences were sampled for the training of the NPC.

4.8 Conclusion and Future Work

We introduced a MARL problem with a rule-based player and RL-based NPCs. We then presented the FAILMAKER-ADVRL algorithm, which trains NPCs in an adversarial but natural manner. By using the techniques of CI, AdvRA, and PS-RBP, FAILMAKER-ADVRL efficiently trains adversarial NPCs. We demonstrated the effectiveness of FAILMAKER-ADVRL through two types of experiments including one using a 3D autonomous driving simulator.

As future work, it would be an interesting direction to consider how to create adversarial situations while incorporating recognition capability of the autonomous vehicles [80, 38].

4.9 Limitation

The limitations of the current algorithm are as follows. First, we achieve only the similar failure scenario to the ones that humans could easily create. For example, failure-scenarios by FAILMAKER-ADVRL could be created by making the adversarial NPCs get close the player. In order to making our algorithm more useful in real applications, we need to develop more sophisticated one so that the coverage of the failure-scenario is increased. Second, the decision-making algorithm in the player we used in our experiments is not very intelligent, which is easy to fail. We should try using our FAILMAKER-ADVRL and DIV-FAILMAKER-ADVRL whether they could obtain failure-scenarios of the player with more reliable and well-constructed algorithm.

Algorithm 3 DIV-FAILMAKER-ADVRL

```
 1: for $m = 1$ to $m_{\max}$ do
 2:   for $e = 1$ to $e_{\max}$ do
 3:     Initialize a random process $\mathcal{N}$ for exploring an action
 4:     Obtain the initial state $\boldsymbol{x}$
 5:     // Execution
 6:     for $t = 1$ to $T$ do
 7:       The player executes the action $a_0$ according to the rule $\boldsymbol{\mu}_0$.
 8:       For the adversarial NPC $i$, select the action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$
 9:       NPCs execute the actions $a = (a_1, \ldots, a_N)$ and get their personal reward $r$ and new
          state $\boldsymbol{x}'$
10:       Calculate KVfD and store $(\boldsymbol{x}, a, r, \boldsymbol{x}', b)$ in temporal replay buffer $\mathcal{D}_{\text{tmp}}$
11:       $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
12:     end for
13:     // Experience Storing with Filtration (ESwF)
14:     if the player succeeds (i.e., NPCs fail) then
15:       Samples in $\mathcal{D}_{\text{tmp}}$ are moved to $\mathcal{D}$ with a probability of $p_1$
16:     else
17:       if a similar failure-scenario is in $\Lambda_{m-1}$ then
18:         Reward is set as $-c$
19:         Samples in $\mathcal{D}_{\text{tmp}}$ are moved to $\mathcal{D}$ with a probability of $p_2$
20:       else
21:         Samples in $\mathcal{D}_{\text{tmp}}$ are moved to $\mathcal{D}$ with a probability of $p_3$
22:       end if
23:     end if
24:     // Training of the NPCs
25:     for NPC $i = 1$ to $N$ do
26:       Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\boldsymbol{x}', a'_0, a'_1, \ldots, a'_N)|_{a'_0=\boldsymbol{\mu}'_0(o_0^j), a'_l=\boldsymbol{\mu}'_l(o_l^j)}$
27:       Update critic using the following loss function $\mathcal{L}(\theta_i) =$
          $\frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}^j, a_0^j, a_1^j, \ldots, a_N^j)\right)^2$
28:       Update actor by:
          $\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}^j, a_0^j, a_1^j, \ldots, a_N^j)|_{a_0=\boldsymbol{\mu}_0(o_0^j), a_i=\boldsymbol{\mu}_i(o_i^j)}$
29:     end for
30:     Update the target network parameters: $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
31:   end for
32:   Get registered KVfDs (Reg-KVfDs) $\lambda_m$, and update the set by $\Lambda_m = \lambda_1 \cup \ldots \cup \lambda_m$
33:   Transfer $\mathcal{D}$ to $(m+1)$-th generation NPCs after deleting the experiences with $\lambda_m$
34: end for
```

Chapter 5

Conclusion

In this dissertation, we have studied how to utilize reinforcement learning for achieving safe, reliable systems.

In Chapter 3, we have presented a safe reinforcement learning algorithm called SNO-MDP. This algorithm theoretically guarantees near-optimality and safety not only during inference but also during training by (conservatively) learning the structure of reward and safety functions using GPs. Empirically, SNO-MDP achieves better performance in terms of the cumulative reward while guaranteeing safety with high probability. As an additional contribution, the experiment was conducted using a newly developed simulation environment called GP-Safety-Gym, which has been open-sourced in GitHub.

In Chapter 4, we then introduced a MARL algorithm for testing a rule-based player by training RL-based NPCs in an adversarial way. Our algorithm is called FailMaker-AdvRL, which trains NPCs in an adversarial yet natural manner. Our techniques called CI, AdvRA, and PS-RBP enable us to train adversarial NPCs efficiently. We demonstrated the effectiveness of our method through two types of experiments including one using a 3D autonomous driving simulator, AirSim. Also, we proposed Div-FailMaker-AdvRL for obtaining diverse failure-scenarios of the tested agent.

5.1 Future Work

There are several possible direction as future work.

First, our safe RL algorithm works for tabular MDPs; hence, we cannot apply it to continuous control problems. Many safety-critical RL problems (e.g., robot control, autonomous driving) are essentially associated with continuous control. Therefore, it would be an interesting direction to consider safe RL algorithm which can be applied in continuous state-action spaces. Given its tremendous success especially in the large state-action spaces, deep RL algorithms will be more and more reasonable solutions in real problems. Our current safe RL algorithms are based on traditional method (e.g., value iteration, policy iteration), but it would be better if we could incorporate safety in deep RL frameworks for achieving a policy to cope with more complicated and difficult environments.

Second, our proposed algorithm, SNO-MDP is based on regularity and Lipschitz assumptions, which are not true in many real applications. Hence, to develop an algorithm useful in real-world problems, we need to remove the above strong assumptions. In this case, it is reasonable to assume that the agent can observe the multiple states of its neighbors so that the assumptions on the properties of the safety function are not required.

Third, to develop a safe RL algorithm which is practically useful in real problems such as autonomous driving or autonomous flight systems, it is required to be applicable with multi-agent systems. It is important to develop safe RL algorithms which enable agents to behave properly in multi-agent environments.

Fourth, it will be important to apply our method to more realistic environments that include pedestrians or traffic signs. The first step to accomplishing this would be to craft an adversarial environment (e.g., the shape of the intersection). Also, in this work, we do not consider computer vision aspects. It will be significant to create an integrated adversarial situation while incorporating perception capabilities.

Finally, in order to apply method (either policy development and testing) in real systems, explainability and interpretability are essential requirements. It would be significant direction to consider how to get the policy of the agent in some explainable form of representation.

References

[1] Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International Conference on Machine Learning (ICML)*.

[2] Altman, E. (1999). *Constrained Markov decision processes*, volume 7. CRC Press.

[3] Araya, M., Buffet, O., and Thomas, V. (2012). Near-optimal BRL using optimistic local transitions. In *International Conference on Machine Learning (ICML)*.

[4] Artzi, S., Dolby, J., Jensen, S. H., Møller, A., and Tip, F. (2011). A framework for automated testing of javascript web applications. In *International Conference on Software Engineering (ICSE)*, pages 571–580.

[5] Aswani, A., Gonzalez, H., Sastry, S. S., and Tomlin, C. (2013). Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226.

[6] Auer, P. and Ortner, R. (2007). Logarithmic online regret bounds for undiscounted reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.

[7] Bäuerle, N. and Ott, J. (2011). Markov decision processes with average-value-at-risk criteria. *Mathematical Methods of Operations Research*, 74(3):361–379.

[8] Bauersfeld, S. and Vos, T. (2012). A reinforcement learning approach to automated gui robustness testing. In *Fast abstracts of the 4th symposium on search-based software engineering (SSBSE 2012)*, pages 7–12.

[9] Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Neural Information Processing Systems (NeurIPS)*.

[10] Biyik, E., Margoliash, J., Alimo, S. R., and Sadigh, D. (2019). Efficient and safe exploration in deterministic markov decision processes with unknown transition models. In *2019 American Control Conference (ACC)*, pages 1792–1799. IEEE.

[11] Bodnar, C., Li, A., Hausman, K., Pastor, P., and Kalakrishnan, M. (2019). Quantile qt-opt for risk-aware vision-based robotic grasping. *arXiv preprint arXiv:1910.02787*.

[12] Brafman, R. I. and Tennenholtz, M. (2002). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, pages 213–231.

[13] Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*. Springer Science & Business Media.

[14] Cannon, M. and Kouvaritakis, B. (2005). Optimizing prediction dynamics for robust mpc. *IEEE Transactions on Automatic Control*, 50(11):1892–1897.

[15] Chow, Y., Nachum, O., Faust, A., Duenez-Guzman, E., and Ghavamzadeh, M. (2019). Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*.

[16] Chowdhury, S. R. and Gopalan, A. (2017). On kernelized multi-armed bandits. In *International Conference on Machine Learning (ICML)*.

[17] Cowen-Rivers, A. I., Palenicek, D., Moens, V., Abdullah, M., Sootla, A., Wang, J., and Ammar, H. (2020). Samba: Safe model-based & active reinforcement learning. *arXiv preprint arXiv:2006.09436*.

[18] Devlin, S., Yliniemi, L., Kudenko, D., et al. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *International Conference on Autonomous agents and Multi-Agent Systems (AAMAS)*.

[19] Doyle, J. (1996). Robust and optimal control. In *Proceedings of 35th IEEE Conference on Decision and Control*, volume 2, pages 1595–1598. IEEE.

[20] Eysenbach, B., Gu, S., Ibarz, J., and Levine, S. (2018). Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*.

[21] Fisac, J. F., Akametalu, A. K., Zeilinger, M. N., Kaynama, S., Gillula, J., and Tomlin, C. J. (2018). A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*.

[22] Fleming, W. H. and McEneaney, W. M. (1995). Risk-sensitive control on an infinite time horizon. *SIAM Journal on Control and Optimization*, 33(6):1881–1915.

[23] Foerster, J., Farquhar, G., Afouras, T., et al. (2018). Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence (AAAI)*.

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

[25] Green, M. and Limebeer, D. J. (2012). *Linear robust control*. Courier Corporation.

[26] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 1.

[27] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[28] Harries, L., Clarke, R. S., Chapman, T., Nallamalli, S. V., Ozgur, L., Jain, S., Leung, A., Lim, S., Dietrich, A., Hernández-Lobato, J. M., et al. (2020). Drift: Deep reinforcement learning for functional software testing. *arXiv preprint arXiv:2007.08220*.

[29] Horgan, D., Quan, J., Budden, D., et al. (2018). Distributed prioritized experience replay. In *International Conference on Learning Representation (ICLR)*.

[30] Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive markov decision processes. *Management science*, 18(7):356–369.

[31] Kakade, S. M. et al. (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England.

[32] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR.

[33] Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.

[34] Khalil, H. K. and Grizzle, J. W. (2002). *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ.

[35] Kim, H. J. and Shim, D. H. (2003). A flight control system for aerial robots: algorithms and experiments. *Control engineering practice*, 11(12):1389–1400.

[36] Kolter, J. Z. and Ng, A. Y. (2009). Near-Bayesian exploration in polynomial time. In *International Conference on Machine Learning (ICML).*

[37] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014. Citeseer.

[38] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

[39] Lample, G. and Chaplot, D. S. (2017). Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[40] Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *International Conference on Machine Learning (ICML).*

[41] Levine, S., Finn, C., Darrell, T., et al. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research (JMLR)*, 17(1):1334–1373.

[42] Li, S., Wu, Y., Cui, X., et al. (2019). Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *AAAI Conference on Artificial Intelligence (AAAI).*

[43] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML).*

[44] Lowe, R., Wu, Y., Tamar, A., et al. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS).*

[45] Lütjens, B., Everett, M., and How, J. P. (2019). Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE.

[46] Marchetto, A. and Tonella, P. (2011). Using search-based algorithms for ajax event sequence generation during testing. *Empirical Software Engineering*, 16(1):103–140.

[47] Marcus, S. I., Fernández-Gaucherand, E., Hernández-Hernandez, D., Coraluppi, S., and Fard, P. (1997). Risk sensitive markov decision processes. In *Systems and control in the twenty-first century*, pages 263–279. Springer.

[48] McEwen, A. S., Eliason, E. M., Bergstrom, J. W., et al. (2007). Mars reconnaissance orbiter's high resolution imaging science experiment (HiRISE). *Journal of Geophysical Research: Planets*, 112(E5).

[49] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

[50] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

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

[52] Montgomery, D. C., Peck, E. A., and Vining, G. G. (2021). *Introduction to linear regression analysis*. John Wiley & Sons.

[53] Mordatch, I. and Abbeel, P. (2017). Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*.

[54] Nguyen, D. T., Kumar, A., and Lau, H. C. (2018). Credit assignment for collective multiagent RL with global rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[55] Nonnengart, A., Klusch, M., and Müller, C. (2019). Crisgen: Constraint-based generation of critical scenarios for autonomous vehicles. In *International Symposium on Formal Methods*, pages 233–248. Springer.

[56] Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

[57] Pei, K., Cao, Y., Yang, J., et al. (2017). Deepxplore: Automated whitebox testing of deep learning systems. In *Symposium on Operating Systems Principles (SOSP)*.

[58] Pinto, L., Davidson, J., Sukthankar, R., et al. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[59] Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer.

[60] Rausand, M. (2014). *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons.

[61] Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking safe exploration in deep reinforcement learning.

[62] Schaul, T., Quan, J., Antonoglou, I., et al. (2016). Prioritized experience replay. In *International Conference on Learning Representation (ICLR).*

[63] Schölkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press.

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

[65] Schwarm, A. T. and Nikolaou, M. (1999). Chance-constrained model predictive control. *AIChE Journal*, 45(8):1743–1752.

[66] Seber, G. A. and Lee, A. J. (2012). *Linear regression analysis*, volume 329. John Wiley & Sons.

[67] Shah, S., Dey, D., Lovett, C., et al. (2017). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics (FSR).*

[68] Silver, D., Huang, A., Maddison, C. J., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484.

[69] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

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

[71] Singh, S., Chow, Y., Majumdar, A., and Pavone, M. (2018). A framework for time-consistent, risk-sensitive model predictive control: Theory and algorithms. *IEEE Transactions on Automatic Control*, 64(7):2905–2912.

[72] Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML).*

[73] Stanton, N. A. and Young, M. S. (1998). Vehicle automation and driving performance. *Ergonomics*, 41(7):1014–1028.

[74] Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). PAC model-free reinforcement learning. In *International Conference on Machine Learning (ICML).*

[75] Strehl, A. L. and Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In *International Conference on Machine Learning (ICML).*

[76] Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

[77] Sui, Y., Gotovos, A., Burdick, J. W., and Krause, A. (2015). Safe exploration for optimization with Gaussian processes. In *International Conference on Machine Learning (ICML)*.

[78] Sui, Y., Zhuang, V., Burdick, J. W., and Yue, Y. (2018). Stagewise safe Bayesian optimization with Gaussian processes. In *International Conference on Machine Learning (ICML)*.

[79] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

[80] Szegedy, C., Zaremba, W., Sutskever, I., et al. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

[81] Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *International Conference on Software Engineering (ICSE)*.

[82] Turchetta, M., Berkenkamp, F., and Krause, A. (2016). Safe exploration in finite Markov decision processes with Gaussian processes. In *Neural Information Processing Systems (NeurIPS)*.

[83] Turchetta, M., Berkenkamp, F., and Krause, A. (2019). Safe exploration for interactive machine learning. In *Neural Information Processing Systems (NeurIPS)*.

[84] Turchetta, M., Kolobov, A., Shah, S., Krause, A., and Agarwal, A. (2020). Safe reinforcement learning via curriculum induction. *arXiv preprint arXiv:2006.12136*.

[85] Urmson, C., Anhalt, J., Bagnell, D., et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466.

[86] Vogel, K. (2003). A comparison of headway and time to collision as safety indicators. *Accident analysis & prevention*, 35(3):427–433.

[87] Wachi, A., Sui, Y., Yue, Y., and Ono, M. (2018). Safe exploration and optimization of constrained MDPs using Gaussian processes. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[88] Wang, J., Pun, A., Tu, J., Manivasagam, S., Sadat, A., Casas, S., Ren, M., and Urtasun, R. (2021). Advsim: Generating safety-critical scenarios for self-driving vehicles. *arXiv preprint arXiv:2101.06549*.

[89] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[90] Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784. PMLR.

[91] Wilson, A. G., Dann, C., and Nickisch, H. (2015). Thoughts on massively scalable gaussian processes. *arXiv preprint arXiv:1511.01870*.

[92] Wolpert, D. H. and Tumer, K. (2002). Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific.

[93] Zhou, K. and Doyle, J. C. (1998). *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ.

Appendix A

Appendix for Safe RL in Constrained MDPs

A.1. Preliminary Lemma

Lemma 3. *For two arbitrary functions $f_1(x)$ and $f_2(x)$, the following inequality holds:*

$$\max_x f_1(x) - \max_x f_2(x) \geq \min_x (f_1(x) - f_2(x)).$$

Proof. For two arbitrary functions $f_4(x)$ and $f_5(x)$, the following inequality holds:

$$\max_x f_4(x) + \max_x f_5(x) \ \geq \max_x \{f_4(x) + f_5(x)\}.$$

Let $f_2(x) = f_4(x) + f_5(x)$ and $f_3(x) = -f_4(x)$. Then,

$$\begin{aligned} \max_x\{-f_3(x)\} + \max_x\{f_2(x) + f_3(x)\} \ &\geq \max_x f_2(x), \\ \max_x\{f_2(x) + f_3(x)\} - \max_x f_2(x) \ &\geq -\max_x\{-f_3(x)\}, \\ \max_x\{f_2(x) + f_3(x)\} - \max_x f_2(x) \ &= \min_x f_3(x). \end{aligned}$$

Finally, let $f_1(x) = f_2(x) + f_3(x)$. Then, the desired lemma is obtained. $\square$

A.2. Near-optimality

Lemma 4. *Let* $J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)$ *be the value function calculated by* SNO-MDP *without the* ES^2 *algorithm. Then,* $J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)$ *satisfies the following inequality:*

$$J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t).$$

Proof. Consider a state $\boldsymbol{s}_t$ and beliefs $\boldsymbol{b}^r_t$ and $\boldsymbol{b}^g_t$. Also, let I denote the following safety indicator function:

$$I(\boldsymbol{s}) := \begin{cases} 1 & \text{if } \boldsymbol{s} \in \bar{R}_{\epsilon_g}(S_0), \\ 0 & \text{otherwise.} \end{cases} \tag{A.1}$$

Then, the following chain of equations and inequalities holds:

$$\begin{aligned}
& J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_t) \\
= & \max_{s_{t+1} \in \mathcal{X}^-_{t^*}} [\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,] - \max_{s_{t+1} \in \bar{R}_{\epsilon_g}(S_0)} [\, r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \,] \\
\geq & \max_{s_{t+1} \in \bar{R}_{\epsilon_g}(S_0)} [\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,] - \max_{s_{t+1} \in \bar{R}_{\epsilon_g}(S_0)} [\, r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \,] \\
= & \max_{a_t} [\, I(\boldsymbol{s}_{t+1}) \cdot \{U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)\} \,] - \max_{a_t} [\, I(\boldsymbol{s}_{t+1}) \cdot \{r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1})\} \,] \\
\geq & \min_{a_t} [\, I(\boldsymbol{s}_{t+1}) \cdot \{U_t(\boldsymbol{s}_{t+1}) - r(\boldsymbol{s}_{t+1})\} + \gamma I(\boldsymbol{s}_{t+1}) J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - \gamma I(\boldsymbol{s}_{t+1}) V^*(\boldsymbol{s}_{t+1}) \,] \\
= & \min_{a_t} [\, I(\boldsymbol{s}_{t+1}) \cdot \{U_t(\boldsymbol{s}_{t+1}) - r(\boldsymbol{s}_{t+1})\} + \gamma I(\boldsymbol{s}_{t+1}) \{J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_{t+1})\} \,] \,.
\end{aligned}$$

The third line follows from $\mathcal{X}^-_{t^*} \supseteq \bar{R}_{\epsilon_g}(S_0)$ in Theorem 1. Also, the fourth line follows from the definition of I, and the fifth line follows from Lemma 3. Because $\boldsymbol{s}$ is arbitrary in the above derivation, we have

$$\begin{aligned}
& \min_{\boldsymbol{s}_t} [\, J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_t) \,] \\
& \geq \min_{\boldsymbol{s}_{t+1}} [\, I(\boldsymbol{s}_{t+1}) \{U_t(\boldsymbol{s}_{t+1}) - r(\boldsymbol{s}_{t+1})\} + \gamma I(\boldsymbol{s}_{t+1}) \{J^*(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_{t+1})\} \,] \,.
\end{aligned}$$

By Lemma 2, the following equation holds with probability at least $1 - \Delta^r$:

$$\min_{\boldsymbol{s}_t} [\, J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,] \geq \gamma \cdot \min_{\boldsymbol{s}_{t+1}} [I(\boldsymbol{s}_{t+1}) \{J^*_{\mathcal{X}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_{t+1})\} \,]$$

Repeatedly applying this equation proves the desired lemma. Therefore, we have

$$J^*_{\mathcal{X}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t)$$

with high probability. $\square$

Lemma 5. (Generalized induced inequality) *Let $\boldsymbol{b}^r, \boldsymbol{b}^g, r$ and $\hat{\boldsymbol{b}}^r, \hat{\boldsymbol{b}}^g, \hat{r}$ be the beliefs (over reward and safety, respectively) and reward functions (including the exploration bonus) that are identical on some set of states Ω — i.e., $\boldsymbol{b}^r = \hat{\boldsymbol{b}}^r$, $\boldsymbol{b}^g = \hat{\boldsymbol{b}}^g$, and $r = \hat{r}$ for all $\boldsymbol{s} \in \Omega$. Let $P(A_\Omega)$ be the probability that a state not in Ω is generated when starting from state $\boldsymbol{s}$ and following a policy π. If the value is bound in $[0, V_{\max}]$, then*

$$V^\pi(\boldsymbol{s}, \boldsymbol{b}^r, \boldsymbol{b}^g, r) \geq V^\pi(\boldsymbol{s}, \hat{\boldsymbol{b}}_r, \hat{\boldsymbol{b}}_g, \hat{r}) - V_{\max} P(A_\Omega),$$

where we now make explicit the dependence of the value function on the reward.

Proof. The lemma follows from Lemma 8 in Strehl and Littman [75]. $\square$

Lemma 6. *Assume that the reward function r satisfies $\|r\|^2_k \leq B^r$, and that the noise n^r_t is σ_r-sub-Gaussian. If $\alpha_t = B^r + \sigma_r\sqrt{2(\Gamma^r_{t-1} + 1 + \log(1/\Delta^r))}$ and $C_r = 8/\log(1+\sigma_r^{-2})$, then the following holds:*

$$\frac{1}{2}\sqrt{\frac{C_r \alpha_{t^*} \Gamma^r_{t^*}}{t^*}} \geq \alpha^{1/2}_{t^*} \sigma^r_{t^*}(\boldsymbol{s}),$$

with probability at least $1 - \Delta^r$.

Proof. The lemma follows from Lemma 4 in Chowdhury and Gopalan [16]. $\square$

A.3. ES^2 algorithm

Lemma 7. *Assume that $\mathcal{Y}_t \subseteq \mathcal{X}^-_t$ holds. Suppose that we obtain the optimal policy, π^*_y on the basis of $J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) = \max_{\boldsymbol{s}_{t+1} \in \mathcal{Y}_t}\big[U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)\big]$. Then, for all t, the following holds:*

$$\boldsymbol{s}_t \in \mathcal{Y}_t \Longrightarrow \boldsymbol{s}_{t+1} \in \mathcal{Y}_t.$$

Proof. When $\mathcal{Y}_t \subseteq \mathcal{X}^-_t$ holds, we have

$$\begin{aligned}\{\boldsymbol{s}' \in \mathcal{S}^+ \mid \forall \boldsymbol{s} \in \mathcal{Y}_t : \boldsymbol{s}' = f(\boldsymbol{s}, \pi^*_y(a \mid \boldsymbol{s}))\} &\subseteq \{\boldsymbol{s}' \in \mathcal{S}^+ \mid \forall \boldsymbol{s} \in \mathcal{X}^-_t : \boldsymbol{s}' = f(\boldsymbol{s}, \pi^*_y(a \mid \boldsymbol{s}))\} \\ &= \mathcal{Y}_t.\end{aligned}$$

This means that the next state $\boldsymbol{s}_{t+1}$ will be within $\mathcal{Y}_t$ if the agent is in $\mathcal{Y}_t$ and decides the action based on π^*_y. Therefore, we have the desired lemma. $\square$

Lemma 8. *Assume that $\mathcal{Y}_t \subseteq \mathcal{X}_t^-$ holds, and let $J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)$ be the value function calculated by* SNO-MDP *with the* ES^2 *algorithm. Then, for all $\boldsymbol{s}_t \in \mathcal{X}_t^-$, $J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)$ satisfies the following equation:*

$$J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t).$$

Proof. Consider a state $\boldsymbol{s}_t \in \mathcal{X}_t^-$ and beliefs $\boldsymbol{b}^r$ and $\boldsymbol{b}^g$. Also, we define the function I as in (A.1). Then, the following chain of the equations and inequalities holds:

$$\begin{aligned}
& J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_t) \\
&= \max_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,\big] - \max_{a_t} \big[\, I(\boldsymbol{s}_{t+1}) \cdot \{ r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \} \,\big] \\
&= \max_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,\big] - \max_{s_{t+1} \in \mathcal{X}_t^+} \big[\, I(\boldsymbol{s}_{t+1}) \cdot \{ r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \} \,\big] \\
&= \max_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \,\big] - \max_{s_{t+1} \in \mathcal{Y}_t} \big[\, I(\boldsymbol{s}_{t+1}) \cdot \{ r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \} \,\big] \\
&\geq \min_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - I(\boldsymbol{s}_{t+1}) \cdot \{ r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \} \,\big] \\
&\geq \min_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - \{ r(\boldsymbol{s}_{t+1}) + \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \} \,\big] \\
&= \min_{s_{t+1} \in \mathcal{Y}_t} \big[\, U_t(\boldsymbol{s}_{t+1}) - r(\boldsymbol{s}_{t+1}) + \gamma J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - \gamma V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \,\big].
\end{aligned}$$

The second and third lines follow from the definitions of I and $V^*_{\mathcal{M}}$. The forth line follows from the definition of $\mathcal{Y}$ and the assumption of $\mathcal{Y}_t \subseteq \mathcal{X}_t^-$. The fifth line follows from Lemma 3.

Then, by Lemma 2, the following equation holds with probability at least $1 - \Delta^r$:

$$\begin{aligned}
\min_{s_t \in \mathcal{X}_t^-} \big[\, J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*(\boldsymbol{s}_t) \} \,\big] &\geq \gamma \cdot \min_{s_{t+1} \in \mathcal{Y}_t} \big[\, J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+1}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*_{\mathcal{M}}(\boldsymbol{s}_{t+1}) \,\big] \\
&\geq \gamma^2 \cdot \min_{s_{t+2} \in \mathcal{Y}_t} \big[\, J^*_{\mathcal{Y}}(\boldsymbol{s}_{t+2}, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^*_{\mathcal{M}}(\boldsymbol{s}_{t+2}) \,\big].
\end{aligned}$$

The second line follows from Lemma 7. Repeatedly applying this equation proves the desired lemma. Therefore, for all $\boldsymbol{s}_t \in \mathcal{X}_t^-$, we have

$$J^*_{\mathcal{Y}}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t).$$

□

A.4. Main Theoretical Results

Theorem 1. *Assume that the safety function g satisfies $\|g\|_k^2 \leq B^g$ and is L-Lipschitz continuous. Also, assume that $S_0 \neq \emptyset$ and $g(\boldsymbol{s}) \geq h$ for all $\boldsymbol{s} \in S_0$. Fix any $\epsilon_g > 0$ and $\Delta^g \in (0, 1)$. Suppose that we conduct the stage of "exploration of safety" with the noise n_t^g being σ_g-sub-Gaussian, and that $\beta_t = B^g + \sigma_g\sqrt{2(\Gamma_{t-1}^g + 1 + \log(1/\Delta^g))}$ until $\max_{\boldsymbol{s}\in G_t} w_t(\boldsymbol{s}) < \epsilon_g$ is achieved. Finally, let t^* be the smallest integer satisfying*

$$\frac{t^*}{\beta_{t^*}\Gamma_{t^*}^g} \geq \frac{C_g|\bar{R}_0(S_0)|}{\epsilon_g^2} \cdot D(\mathcal{M}),$$

with $C_g = 8/\log(1 + \sigma_g^{-2})$. Then, the following statements jointly hold with probability at least $1 - \Delta^g$:

- $\forall t \geq 1$, $g(\boldsymbol{s}_t) \geq h$,
- $\exists t_0 \leq t^*$, $\bar{R}_{\epsilon_g}(S_0) \subseteq \mathcal{X}_{t_0}^- \subseteq \bar{R}_0(S_0)$.

Proof. This is an extension of Theorem 1 in Turchetta et al. [82] to our settings, where t represents not the number of samples but the number of actions. □

Theorem 2. *Assume that the reward function r satisfies $\|r\|_k^2 \leq B^r$, and that the noise is σ_r-sub-Gaussian. Let π_t denote the policy followed by* SNO-MDP *at time t, and let $\boldsymbol{s}_t$ and $\boldsymbol{b}_t^r, \boldsymbol{b}_t^g$ be the corresponding state and beliefs, respectively. Let t^* be the smallest integer satisfying $\frac{t^*}{\beta_{t^*}\Gamma_{t^*}^g} \geq \frac{C_g|\bar{R}_0(S_0)|}{\epsilon_g^2} D(\mathcal{M})$, and fix any $\Delta^r \in (0, 1)$. Finally, set $\alpha_t = B^r + \sigma_r\sqrt{2(\Gamma_{t-1}^r + 1 + \log(1/\Delta^r))}$ and*

$$\epsilon_V^* = V_{\max} \cdot (\Delta^g + \Sigma_{t^*}^r / R_{\max}),$$

with $\Sigma_{t^*}^r = \frac{1}{2}\sqrt{\frac{C_r \alpha_{t^*}\Gamma_{t^*}^r}{t^*}}$. Then, with high probability,

$$V^{\pi_t}(\boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) \geq V^*(\boldsymbol{s}_t) - \epsilon_V^*$$

— i.e., the algorithm is ϵ_V^*-close to the optimal policy — for all but t^* time steps, while guaranteeing safety with probability at least $1 - \Delta^g$.

Proof. Define $\tilde{r}$ as the reward function (including the exploration bonus) that is used by SNO-MDP. Let $\hat{r}$ be a reward function equal to r on Ω and equal to $\tilde{r}$ elsewhere. Furthermore, let $\tilde{\pi}$ be the policy followed by SNO-MDP at time t, that is, the policy calculated on the basis of the current beliefs, (i.e., $\boldsymbol{b}_t^r$ and $\boldsymbol{b}_t^g$) and the reward $\tilde{r}$. Finally, let A_Ω be the event in which $\tilde{\pi}$ escapes from Ω. Then,

$$V^{\pi_t}(r, \boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) \geq V^{\tilde{\pi}}(\hat{r}, \boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) - V_{\max} P(A_\Omega)$$

by Lemma 5. In addition, note that, for all $t \geq t^*$, because $\hat{r}$ and $\tilde{r}$ differ by at most $\alpha_{t^*}^{1/2} \sigma_{t^*}^r$ at each state,

$$\begin{aligned} |V^{\tilde{\pi}}(\hat{r}, \boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g) - V^{\tilde{\pi}}(\tilde{r}, \boldsymbol{s}_t, \boldsymbol{b}_t^r, \boldsymbol{b}_t^g)| &\leq \frac{1}{1-\gamma} \cdot \alpha_{t^*}^{1/2} \sigma_{t^*}^r(\boldsymbol{s}) \\ &\leq V_{\max}/R_{\max} \cdot \Sigma_{t^*}^r. \end{aligned} \tag{A.2}$$

For the above inequality, we used Lemma 6. Here, consider the case of $\Omega = \mathcal{X}_{t^*}^-$. Once the safe region is fully explored, $P(A_\Omega) \leq \Delta^g$ holds after t^* time steps. Then, the following chain of equations and inequalities holds:

$$\begin{aligned} V^{\pi_t}(R, \boldsymbol{s}, \boldsymbol{b}) &\geq V^{\tilde{\pi}}(\hat{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot P(A_\Omega) \\ &= V^{\tilde{\pi}}(\hat{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot P(A_{\mathcal{X}^-}) \\ &\geq V^{\tilde{\pi}}(\hat{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot \Delta^g \\ &\geq V^{\tilde{\pi}}(\tilde{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot (\Delta^g + \Sigma_{t^*}^r / R_{\max}) \\ &= J_{\mathcal{X}}^*(\tilde{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot (\Delta^g + \Sigma_{t^*}^r / R_{\max}) \\ &\geq V^*(R, \boldsymbol{s}) - V_{\max} \cdot (\Delta^g + \Sigma_{t^*}^r / R_{\max}). \end{aligned}$$

In this derivation, the second line follows from the assumption of $\Omega = \mathcal{X}^-$, the third line follows from $P(A_{\mathcal{X}^-}) \leq \Delta^g$, the fourth line follows from (A.2), the fifth line follows from the fact that $\tilde{\pi}$ is precisely the optimal policy for $\tilde{R}$ and $\boldsymbol{b}$, and the final line follows from Lemma 4. $\square$

Theorem 3. *Assume that the reward function r satisfies* $\|r\|_k^2 \leq B^r$*, and that the noise is* σ_r*-sub-Gaussian. Let* π_t *denote the policy followed by* SNO-MDP *with the the* ES^2 *algorithm at time t, and let* $\boldsymbol{s}_t$ *and* $\boldsymbol{b}_t^r, \boldsymbol{b}_t^g$ *be the corresponding state and beliefs, respectively. Let* $\tilde{t}$ *be the smallest integer for which (3.4) holds, and fix any* $\Delta^r \in (0,1)$*. Finally, set* $\alpha_t = B^r + \sigma_r \sqrt{2(\Gamma_{t-1}^r + 1 + \log(1/\Delta^r))}$ *and*

$$\tilde{\epsilon}_V = V_{\max} \cdot (\Delta^g + \Sigma_{\tilde{t}}^r / R_{\max}),$$

with $\Sigma^r_{\tilde{t}} = \frac{1}{2}\sqrt{\frac{C_r \alpha_{\tilde{t}} \Gamma^r_{\tilde{t}}}{\tilde{t}}}$*. Then, with high probability,*

$$V^{\pi_t}(\boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^*(\boldsymbol{s}_t) - \tilde{\epsilon}_V$$

— i.e., the algorithm is $\tilde{\epsilon}_V$*-close to the optimal policy — for all but* $\tilde{t}$ *time steps while guaranteeing safety with probability at least* $1 - \Delta^g$*.*

Proof. The proof of Theorem 3 is analogous to that of Theorem 2. Define $\tilde{r}$ as the reward function (including the exploration bonus) that is used by SNO-MDP. Let $\hat{r}$ be a reward function equal to r on $\mathcal{Y}$ and equal to $\tilde{r}$ elsewhere. Furthermore, let $\tilde{\pi}$ be the policy followed by SNO-MDP with the ES^2 algorithm at time t, that is, the policy calculated on the basis of the current beliefs, (i.e., $\boldsymbol{b}^r_t$ and $\boldsymbol{b}^g_t$) and the reward $\tilde{r}$. Finally, let $A_\mathcal{Y}$ be the event in which $\tilde{\pi}$ escapes from $\mathcal{Y}$. Then,

$$V^{\pi_t}(r, \boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) \geq V^{\tilde{\pi}}(\hat{r}, \boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V_{\max} P(A_\mathcal{Y})$$

by Lemma 5. In addition, note that, for all $t \geq \tilde{t}$, because $\hat{r}$ and $\tilde{r}$ differ by at most $\alpha^{1/2}_{\tilde{t}} \sigma^r_{\tilde{t}}$ at each state,

$$\begin{aligned} |V^{\tilde{\pi}}(\hat{r}, \boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t) - V^{\tilde{\pi}}(\tilde{r}, \boldsymbol{s}_t, \boldsymbol{b}^r_t, \boldsymbol{b}^g_t)| &\leq \frac{1}{1-\gamma} \cdot \alpha^{1/2}_{\tilde{t}} \sigma^r_{\tilde{t}}(\boldsymbol{s}) \\ &\leq V_{\max}/R_{\max} \cdot \Sigma^r_{\tilde{t}}. \end{aligned} \tag{A.3}$$

For the above inequalities, we used Lemma 6. Then, the following chain of equations and inequalities holds:

$$\begin{aligned} V^{\pi_t}(R, \boldsymbol{s}, \boldsymbol{b}) &= V^{\tilde{\pi}}(\hat{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot P(A_\mathcal{Y}) \\ &\geq V^{\tilde{\pi}}(\hat{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot \Delta^g \\ &\geq V^{\tilde{\pi}}(\tilde{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot (\Delta^g + \Sigma^r_{\tilde{t}}/R_{\max}) \\ &= J^*_\mathcal{Y}(\tilde{R}, \boldsymbol{s}, \boldsymbol{b}) - V_{\max} \cdot (\Delta^g + \Sigma^r_{\tilde{t}}/R_{\max}) \\ &\geq V^*(R, \boldsymbol{s}) - V_{\max} \cdot (\Delta^g + \Sigma^r_{\tilde{t}}/R_{\max}). \end{aligned}$$

In this derivation, the second line follows from $P(A_\mathcal{Y}) \leq \Delta^g$, the third line follows from (A.3), the fourth line follows from the fact that $\tilde{\pi}$ is precisely the optimal policy for $\tilde{R}$ and $\boldsymbol{b}$, and the final line follows from Lemma 8. □