Meta-Optimization in Safe Reinforcement Learning: Enhancing Safety at Training and Deployment with Fewer Hyperparameters

by

Homayoun Honari
Bachelor of Science, Sharif University of Technology, 2022

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Mechanical Engineering

Meta-Optimization in Safe Reinforcement Learning: Enhancing Safety at Training and Deployment with Fewer Hyperparameters

by

Homayoun Honari

Bachelor of Science, Sharif University of Technology, 2022

Supervisory Committee

Dr. Homayoun Najjaran, Supervisor

(Department of Mechanical Engineering)

Dr. Yang Shi, Departmental Member

(Department of Mechanical Engineering)

ABSTRACT

Reinforcement learning (RL) is a trial-and-error framework for enabling intelligent systems to learn the optimal behaviour based on the feedback from the environment. In recent years, successful application of RL in controlling various embodied systems have been observed. However, the real-world deployment and training of RL methods require paying attention to certain limitations imposed by the robot and its surroundings. To address these limitations, safe RL algorithms aim to define safety constraints based on the physics of the system and modify the training regime of the RL methods to satisfy them during training and inference. While safe RL offers a promising direction for achieving real-world deployability, challenges such as sample efficiency and hyperparameter tuning hinders its applicability in real-world scenarios.

To address these challenges, this thesis proposes various approaches. First, a metagradient-based training pipeline called Meta Soft Actor-Critic Lagrangian (Meta SAC-Lag) is proposed which aims to optimize the aforementioned safety-related hyperparameters under the conventional Lagrangian framework. To study the performance, the proposed method is evaluated in various safety-critical simulated environments. In addition, a real-world task is designed, and the algorithm is successfully deployed on a Kinova Gen3 robotic arm to showcase its real-world deployability with minimal hyperparameter tuning requirements. Furthermore, a multi-objective policy optimization framework is proposed which specifies the trade-off between optimality and safety directly and optimizes both of them simultaneously. The competitive performance of the proposed algorithm compared to the state-of-the-art safe RL methods with fewer hyperparameters showcases its potential in providing a powerful alternative framework for safe RL.

Table of Contents

List of Tables

List of Figures

LAY SUMMARY

This thesis explores the enhancement of safety in reinforcement learning algorithms. Safe reinforcement learning is one of the promising approaches for achieving reliable and safe methods suitable for deployment to attain intelligent robots capable of learning in the real-world. The challenges of hyperparameter tuning and sample efficiency are among the most important causes for the deployment of these methods on real control systems. First, a unified approach based on meta-gradient optimization is proposed to optimize the safety-related hyperparameters in the most common formulation of safe RL (Lagrangian optimization). The approach addresses two important challenges of automatic constraint threshold adjustment and safe exploration. Furthermore, in addition to evaluations in standard simulation tasks, a real-world task of pouring coffee is designed and the algorithm is trained on a Kinova Gen3 robotic arm. Second, a multi-objective policy optimization algorithm for optimizing the objectives of reward and safety simultaneously is presented which directly specifies the trade-off between the two objectives. The algorithm provides better or competitive performance in both aspects of safety and optimality compared to the Lagrangian formulation with fewer hyperparameters allowing for better deployability. In summary, the aim of this thesis is to provide safe reinforcement learning algorithms capable of performing favorbly with minimal safety-related hyperparameter tuning requirements.

PREFACE

The contents of this thesis have been developed in the Advanced Control and Intelligent Systems (ACIS) Laboratory at the University of Victoria supported by Apera AI and Mathematics of Information Technology and Complex Systems (MITACS) under IT16412 Mitacs Accelerate. Chapter 3 consists of the material accepted for publication at the proceedings of the *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2024)*. A portion of Chapter 4 includes the material accepted for publication at the *IEEE International Conference on Robotics and Automation (ICRA 2024)*. The complete content of Chapter 4 is being prepared for submission to a journal. In summary, the publication this thesis is based on are:

- **Honari, H.**, Soufi, A.M., Tamizi, M. G., & Najjaran, H. (2024). Meta SAC-Lag: Towards Deployable Safe Reinforcement Learning via MetaGradient-based Hyperparameter Tuning . In 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE.

- **Honari, H.**, Tamizi, M. G., & Najjaran, H. (2024). Safety Optimized Reinforcement Learning via Multi-Objective Policy Optimization. In 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE.

In the aforementioned works, I, Homayoun Honari, was responsible for theory development, programming implementation, experiment development and writing of the most sections of the manuscript text. Mehran Ghafarian Tamizi was engaged in plotting of the experiments results, manuscript drafting, and Chapter 3 design and development of the real-world task. Amir Mehdi Soufi Enayati was involved in research development of Chapter 3, manuscript drafting, and also Chapter 3 design and development of the real-world task. Homayoun Najjaran participated in supervision at every stage of the research and manuscript revision.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Homayoun Najjaran, for his unwavering trust and support throughout this journey. His grant of freedom and continual encouragement has profoundly shaped my development as a researcher.

Furthermore, I would like to express my sincere appreciation to the Apera AI and Mathematics of Information Technology and Complex Systems (MITACS) for supporting me financially under the IT16412 Mitacs Accelerate. In particular, I sincerely appreciate the constant help and guidance provided by Aleksey Nozdryn-Plotnicki from Apera AI.

Next, I give a special thank to Mehran Ghafarian Tamizi. As a friend and colleague, he was besides me throughout this journey and I am forever grateful for his help and support.

Also, I extend my gratitude to Amir Mehdi Soufi Enayati another one of my friends and colleagues. Learning from his knowledge and dedication to work has been inspirational to me.

Finally, I thank all the lab members in the Advanced Control and Intelligent Systems Laboratory (ACIS) for creating a vibrant and supportive environment suitable for doing impactful research.

DEDICATION

I dedicate this thesis to my family. My mom for giving me the emotional support I needed to go through this otherwise impossible journey. My dad for his calmness and support which brought me comfort and peace. Last but not the least, my sister for being an infinite source of happiness and positivity which warmed my heart and made me more hopeful for the future.

Finally, I also dedicate this research to my mentor, Mr. Alireza Shabani. His presence and guidance throughout all these years has opened up my mind to a world otherwise unknown to me and made me become a better person and researcher.

Chapter 1

Introduction

Reinforcement Learning (RL) is one of the major paradigms in Machine Learning (ML) that enables decision-making agents to interact with the environment and make better decisions through trial and error. Compared with other ML sub-fields, RL can bring the advantage of online data gathering and unsupervised policy learning. For that reason, RL has shown its great potential to learn the control process of complex robotics systems, even when there is limited information about its dynamics. However, it is generally known that using RL methods require extensive computational resources and thousands, if not millions, of interactions with the environment in order to converge to a well-performing policy. For that reason, conventionally, the deployment of RL algorithms on real-world systems either involves creating a digital twin of the environment for training the model and then performing Sim2Real transfer to the real-world task or gathering data using an oracle policy and training the policy using offline RL methods. In addition to that, many of the control systems implemented in the real-world have certain constraints that the user would need to make sure are not violated during the operation to have the system work as expected. For example, in autonomous driving tasks avoiding collision with other vehicles and not falling over

the cliff are hard constraints that must be satisfied at all times. In another case, many of the human-robot collaborative scenarios involve executing a task with the help of a human while adhering to some soft constraints. The constraints can include avoiding the exertion of excessive forces on the human or preventing extreme accelerations that could cause distress, ensuring the safety and comfort of the human partner. For those reasons, safe RL methods aim to close the gap by incorporating the constraints into the optimization process of the policy in the same online data gathering fashion and developing more sample-efficient algorithms, especially in terms of the safety constraints, to enable the reliable training and deployment of RL methods on real-world control systems.

1.1 Motivation

While the safe RL field have shown its potential in providing algorithms reliable enough for deployment on a real system, the conventional safety formulation in the RL framework poses several challenges. This research focuses on addressing the following primary challenges.

1.1.1 Hyperparameter Tuning

While the algorithms based on the conventional safe RL formulation have performed favorably in many of the safety-critical tasks, the hyperparameters added to the training pipeline require careful tuning in order to achieve the optimal performance. For that reason, the practice of training and deploying these methods on a system involves extensive trial and error to obtain the optimal values for the hyperparameters on the given task; otherwise, the training process might fail or even cause catastrophic damages to the system or the environment. In addition to that, the addition of

safety to the policy objective formulation can decrease the sample-efficiency even more. Therefore, a training session of a safe RL algorithm might require millions of steps of optimization in order to study the performance given a specific set of hyperparameters.

1.1.2 Sample Efficiency

Another aspect of safe RL that hinders its deployability is the sample-efficiency of these methods. Although this issue affects all RL algorithms, it holds particular significance for those designed with safety in mind. This is due the fact that safe RL methods need to provide a fast convergence of the policy to the safe region so that the potential damage to the control system is minimized as much as possible. Therefore, designing safe RL algorithms that provide relatively fast convergence rates can have important results in enabling robots to be capable of operating in the wild.

1.2 Objectives

To address the primary concerns of this research, several objectives are recognized and addressed in this thesis.

1.2.1 Automated Hyperparameter Tuning

As previously discussed, hyperparameter tuning is one of the significant bottlenecks in the process of deployment of safe RL methods. To this end, in Chapter 3, hyperparameter tuning is studied in the context of Lagrangian-based constrained optimization. The Lagrangian framework is the common approach in safe RL methods. The formulation proposed in Chapter 3 provides an algorithm with minimal hyperparameter tuning requirements. Furthermore, in Chapter 4 a safe RL method is developed un-

der the multi-objective policy optimization formulation which allows for fewer number of hyperparameters for specifying the trade-off between optimality and safety. The formulation proposed in Chapter 4 automatically tunes the aforementioned tradeoff based on the training of the policy and optimizes both of the objectives simultaneously.

1.2.2 Optimality and Safety Performance

Another objective of this research is to enhance the performance of the proposed safe RL formulations. Hence, the methods in this research aim create a balanced tradeoff between optimality and safety by designing objective function that address this balance. To this end, the methods presented in this research are studied in both of these aspects.

1.2.3 Benchmarking and Task Diversity

An important aspect of study in safe RL methods is their usefulness in a diverse set of tasks. Demonstrating that a method is capable of solving tasks with various safety formulation can showcase its reliability in being applied in a novel task. To achieve this objective, the methods presented in this research are studied in several embodied environments with various safety topics.

1.2.4 Real-World Deployment

As previously discussed, safe RL methods are designed to be deployed on real-world control systems. To achieve this objective, all the previously discussed objective can have effect. The aim of the real-world experiment is to study the sample efficiency of the method to solve a real-world task with minimal hyperparameter requirements. A favorable performance in the real-world experiment can demonstrate the reliability of

a safe RL method, especially when it is compared with the engineering effort required in the design of the algorithm. Parameters such as the value of the hyperparameters and the design of the reward function can have significant impact on the performance of the algorithm.

1.3 Contributions

To address the discussed objectives, this thesis provides various solutions in the context of safe RL. To this end, the contributions in this thesis can be categorized as:

1. **MetaGradient-based Hyperparameter Optimization**: In order to perform automated hyperparameter tuning, this thesis proposes the use of meta-gradient optimization. This process can reduce the effect of initial hyperparameter value on the overall performance if the algorithm. The meta-optimization approaches discussed in this thesis aim to propose meta-objectives suitable for use in the context of safe RL. In Chapter 3 a unified meta-gradient based pipeline is proposed that aims to optimize safety-related hyperparameters to address automatic constraint threshold adjustment and safe exploration in the context of Lagrangian safe RL. Furthermore, Chapter 4 provides a constrained meta-gradient formulation to optimization the safety-related hyperparameter which specified the trade-off between reward and safety with safety guarantee of the algorithm.

2. **Multi-Objective Policy Optimization**: In addition to proposing meta-gradient approaches for automated hyperparameter tuning, this thesis proposes an alternative framework for safe RL. In contrast to the conventional safe RL framework, namely Lagrangian Optimization, Chapter 4 proposes a multi-objective policy optimization framework to directly specify the trade-off

between optimality and safety. The use of this framework can provide the advantage of competitive (or better) overall performance with fewer safety-related hyperparameters.

3. **Safety Benchmark Evaluation**: To evaluate the performance of safe RL methods, several simulated environments with various safety topics are used. In Chapter 3 the algorithms are studied in five simulation environments with three main safety topic of locomotion, obstacle avoidance, and manipulation. In Chapter 4 the methods are deployed in seven simulation tasks with the safety topics of system-level safety, collision avoidance, and safe manipulation. The tasks and safety topics are designed such that they cover various definitions of robot dynamics and safety design.

4. **Real-World Task**: In addition to the simulation environments, the Lagrangian method proposed in Chapter 3 is deployed and trained on a real-world robotic arm system to pour coffee into a cup. The aim of the safety task is to pour coffee into the mug as fast as possible while avoiding the spillage of coffee on the table and collision with the objects in environment.

1.4 Thesis Outline

In Chapter 2 the basic related concepts in RL, safe RL, and meta-gradient RL are explained. Moreover, the related literature in the context of this thesis are discussed. This thesis proposes two main safe RL methodologies. Chapter 3 focuses on developing a unified pipeline for optimizing the safety-related hyperparameters in the conventional Lagrangian safe RL formulation under the Soft Actor-Critic architecture. Furthermore, Chapter 4 proposes a multi-objective policy optimization framework for directly specifying the trade-off between optimality and safety with

the guarantee of the safety of the final converged policy. Moreover, the use of a constrained meta-gradient optimization allows the method proposed in the chapter to require minimal hyperparameter tuning. Each chapter contains the discussion of the proposed methodology and extensive study of the performance of the algorithm in various embodied tasks and the comparison methods are performed.

Chapter 2

Background and Preliminaries

In this chapter, the basic concepts and formulations in RL are explained and the related literature important to the scope of this research are discussed.

2.1 Reinforcement Learning

RL [1] provides a trial and error mathematical framework that enables the interaction between the decision-making agent and the environment. In the context of RL, several high-level key concepts are defined:

- **Environment:** The entity with which the agent interacts. The agent chooses and executes its action and the environment would evaluate the action and give the important information back to the agent.
- **State:** The state represents the current position and status of the agent in the environment (e.g., location, velocity, etc.). The state captures all relevant information that the agent needs to make decisions about.
- **Action:** The formulation of the decision in the interaction between the agent and the environment is formulated using the action.

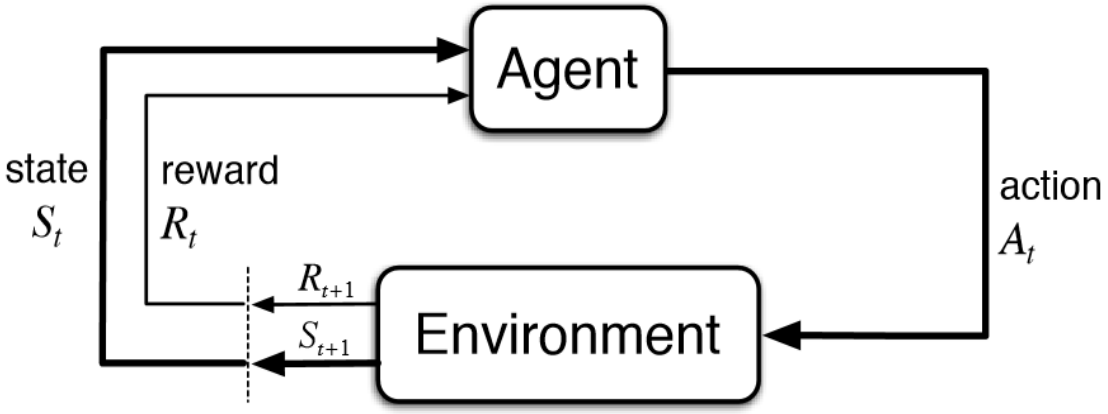


Figure 2.1: Key attributes in the interaction process between a reinforcement learning agent and the environment adopted from [1]

- **Reward:** The reward signal specifies the favorability of the action chosen by the agent in a given state. It is usually engineered to enable the system to exhibit a particular favorable behavior. For example, giving higher rewards for driving in a straight line as fast as possible or winning a game of chess against an opponent.

The interaction process between the agent and the environment is depicted in Figure 2.1. At each step t, the agent receives the state S_t from the environment and chooses the action A_t to be executed in the environment. After the execution, the environment will then evaluate return the reward signal R_{t+1} and the next state of the agent S_{t+1}.

2.1.1 Markov Decision Process

In order to mathematically formulate RL problems, the Markov Decision Process (MDP) framework is used for formalization. The basic definition of MDP is represented as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \rho_0 \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition distribution, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the

reward function, $\gamma \in [0, 1)$ the discount factor, and $\rho_0 : \mathcal{S} \to [0, 1]$ the initial state distribution. In this framework, the Markov property allows for a simplification of the decision-making process by assuming the dependency of the dynamics merely on the latest state and action rather than the whole trajectory:

$$\Pr(s_{t+1}, r_{t+1}|a_t, s_t, a_{t-1}, s_{t-1}, \ldots, a_0, s_0) = \Pr(s_{t+1}, r_{t+1}|a_t, s_t) \tag{2.1}$$

This assumption would make the RL problem more tractable and in most control system applications, with the correct state and action definition, the assumption can be properly met.

2.1.2 Value Function

Value functions in RL serve as measures to evaluate the quality of a policy by estimating the expected sum of rewards it can achieve in any state. The state-action value function is defined as the expected value of the cumulative discounted return of starting from state s, executing action a, and following a specific policy π:

$$Q^\pi(s, a) = \mathbb{E}_{\substack{s_{t+1}\sim\mathcal{P}(s_t,a_t) \\ a_t\sim\pi(s_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a \right] \tag{2.2}$$

where the policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ specifies the action distribution in each state.

Moreover, the state value is specified as the expected value of Equation 2.2 of also following policy π for the starting action:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi} \left[Q^\pi(s, a) \right] = \mathbb{E}_{\substack{s_{t+1}\sim\mathcal{P}(s_t,a_t) \\ a_t\sim\pi(s_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s \right] \tag{2.3}$$

For simplicity in writing, the terms $a_t \sim \pi(s_t)$ and $s_{t+1} \sim \mathcal{P}(s_t, a_t)$ will be reduced to π and $\mathcal{P}$ for the rest of the thesis.

In the context of MDP, and by using the definitions of state and state-action value functions, recursive equations known as Bellman equation can be developed:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}^{\pi}_{\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^{t} r_{t+1} | s_0 = s, a_0 = a\right] \\ &= \mathbb{E}^{\pi}_{\mathcal{P}}\left[r_1 + \sum_{t=1}^{\infty}\gamma^{t} r_{t+1} | s_0 = s, a_0 = a\right] \\ &= \mathbb{E}[r|s_0 = s, a_0 = a] + \gamma \mathbb{E}^{\pi}_{\mathcal{P}}\left[\sum_{t=0}^{\infty}\gamma^{t} r_{t+2} | s_0 = s, a_0 = a\right] \\ &= \mathbb{E}[r|s_0 = s, a_0 = a] + \gamma \mathbb{E}_{s'\sim\mathcal{P}(s,a)}\left[V^{\pi}(s')\right] \\ &= \mathbb{E}[r|s_0 = s, a_0 = a] + \gamma \mathbb{E}_{\substack{s'\sim\mathcal{P}(s,a) \\ a'\sim\pi(s')}}\left[Q^{\pi}(s',a')\right] \end{aligned} \tag{2.4}$$

A similar recursive equation can also be written for the state value function using the definition in Equation 2.3:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{a\sim\pi}\Big[Q^{\pi}(s,a)\Big] \\ &= \mathbb{E}^{\pi}[r|s_0 = s] + \gamma \mathbb{E}_{s'\sim\mathcal{P}(s,\pi(s))}[V(s')] \\ &= \mathbb{E}^{\pi}[r|s_0 = s] + \gamma \mathbb{E}_{\substack{s'\sim\mathcal{P}(s,\pi(s)) \\ a'\sim\pi(s')}}[Q(s',a')] \end{aligned} \tag{2.5}$$

2.1.3 Optimal Policy

The optimal policy in an RL problem is the best performing policy in terms of the cumulative discounted return in every state. To this end, the optimal value function follows:

$$Q^{*}(s,a) = \max_{\pi\in\Pi} Q^{\pi}(s,a),\ \ V^{*}(s) = \max_{a\in\mathcal{A}} Q^{*}(s,a)\ \ \ \forall s \in \mathcal{S} \tag{2.6}$$

where Π is the space of all the possible policies. In order to calculate the optimal value function in a specific RL problem, the bellman optimality equation (similar to Equation 2.4) can be used:

$$Q^*(s,a) = \mathbb{E}[r|s_0 = s, a_0 = a] + \gamma \mathbb{E}_{\substack{s' \sim \mathcal{P}(s,a) \\ a' \sim \pi(s')}} [Q^*(s', a')] \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.7}$$

Using Equation 2.7 provides a system of equations which can be used to solve for every state-action pair in the system. Finally, in any state, the aim is to choose the action with the highest state-action value function. Hence, the optimal policy using Equation 2.6 is specified as:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q^*(s,a) \tag{2.8}$$

2.1.4 Dynamic Programming

As discussed in Section 2.1.3, while Equation 2.7 can theoretically be used to find the optimal value function and the optimal policy, solving the system of equations for the exact values is intractable for most RL applications due to the high dimensionality of the state and/or action spaces, or when these spaces are continuous. Therefore, the dynamic programming method aims to find an approximate solution by iteratively taking an optimization step toward the optimal solution. Basically, dynamic programming is constructed of two stages: policy evaluation and policy improvement. During policy evaluation, at each updating step k, the update rule for the state-action value function can be specified using the Temporal Difference (TD) error:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \beta_Q (r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s,a)) \tag{2.9}$$

where β_Q is the learning rate for updating the value function. After value convergence, the new optimal policy using Equation 2.8 is specified and then the policy is reevaluated. This cycle continues until both the value function and the optimal policy

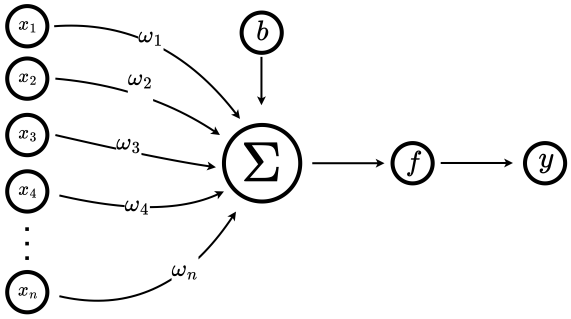


Figure 2.2: Schematic implementation of a neuron in an artificial neural network

have suitably converged to the optimal solution.

2.2 Reinforcement Learning with Function Approximation

Function approximation in RL, conventionally known as Deep RL when using neural networks, aims to address the problem of continuous state or action definition. For that purpose, neural networks are dominantly used as reliable computational function approximation models. Neural networks are bio-inspired computational models built upon the idea of neurons in the brain. A neural network consists of several sequential layers of neurons. As illustrated in Figure 2.2, a neuron takes some signals as input, computes the weighted sum, and applies a nonlinear function(known as the activation function) to compute the output. The neurons in the same layer take as input the same input and their output is given to the following layers of neurons. This process is continued until the the final out layer. Methematically, a neuron layer is represented as an $n \times m$ matrix W where n is the number of input signals and m is the number of neurons in a layer. Hence, the computational rule of each layer taking as input the $n \times 1$ matrix x follows:

$$y = f(W^{\mathrm{T}}x + b) \tag{2.10}$$

where f is a nonlinear activation function and b is an $m \times 1$ bias vector. The activation function enables the model to better capture nonlinear behaviors in the data and its choice can have important impacts. A comprehensive list of the most common activation functions in the literature can be found in [2]. One of the widely-used activation function is the Rectified Linear Unit (ReLU) [3, 4] which is defined as:

$$f(x) = \mathbb{1}[x > 0] \cdot x \tag{2.11}$$

Basically, the activation function is activated whenever the input is greater than zero. Another common activation function used in the literature is Gaussian Error Linear Units (GELU) [5] which is defined as:

$$\begin{aligned} f(x) = x\Pr(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \operatorname{erf}(\frac{x}{\sqrt{2}})\right] \text{ if } X \sim \mathcal{N}(0,1) \\ \approx 0.5x(1 + \tanh\left[\sqrt{2/\pi}(x + 0.044715x^3)\right]) \approx x\sigma(1.702x) \end{aligned} \tag{2.12}$$

Where Φ is the standard Gaussian cumulative distribution function. GELU is designed to provide a smoother function compared to ReLU in order to better capture complex behaviours.

Furthermore, the networks outputs are evaluated using a loss function which compares the ouput of the network with the ground truth. One of the most common loss functions is the mean squared error (MSE) loss function defined as:

$$L(y, \hat{y}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.13}$$

where y and $\hat{y}$ are the $N \times 1$ ground truth and network output vectors. Finally, in order tune the weights of the network according to the data, backpropagation is done. Assuming a one-layer neural network, the weights of the neural network can

be updated by taking the partial derivative of the loss function with respect to the weight matrix:

$$W \leftarrow W + \beta_W \frac{\partial L(y, \hat{y})}{\partial W} = W + \beta_W \frac{\partial L(y, \hat{y})}{\partial \hat{y}} \frac{\partial z}{\partial W} \tag{2.14}$$

where β_W is the weight matrix learning rate and the value of $\hat{y}$ follows Equation 2.10.

2.2.1 Policy Gradient Methods

As discussed in Section 2.1.3, the objective of an RL algorithm is to maximize the expected return of its policy using the measure of value function. Hence, the objective of the algorithm with the policy parameterized as θ is formulated as:

$$\mathcal{J}_\pi(\theta) = \mathbb{E}^{\pi_\theta}_{s\sim\rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] = \int_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \int_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \mathrm{d}a\mathrm{d}s \tag{2.15}$$

Where ρ^π is defined as the discounted state distribution of following policy π:

- $\rho_0(s)$ is the initial state distribution similar to the one discussed in Section 2.1.1.
- $\rho^\pi(s \rightarrow s', k)$: The visitation probability of state s' after moving k steps by following π
- $\rho^\pi(s') = \int_{s\in\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\pi(s \rightarrow s', k) \mathrm{d}s$

The policy gradient theorem [1, 6], shows that the gradient of the objective function w.r.t. the parameter θ can be calculated as:

$$\nabla_\theta \mathcal{J}_\pi(\theta) = \int_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \int_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \mathrm{d}a\mathrm{d}s = \mathbb{E}^{\pi_\theta}_{s\sim\rho^\pi}\left[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(s)\right] \tag{2.16}$$

Using the objective formulation, the policy parameters can then be updated by taking the gradient of the objective function:

$$\theta_{t+1} \leftarrow \theta_t + \beta_\theta \nabla_\theta \mathcal{J}_\pi(\theta_t) \tag{2.17}$$

where β_θ is the learning rate for θ.

2.2.2 Actor-Critic Methods

The policy gradient method in Equation 2.16 provides a powerful formulation to implement RL algorithms. The early algorithms (such as REINFORCE [7]) relied on the Monte Carlo sampling of the returns to estimate the Q values. These methods are called Monte Carlo Policy Gradient algorithms. However, these methods suffer from high variance during the training process. For that reason, later works resulted in estimating the value using another parameterization for the value function. These methods are called actor-critic methods. Parameterizing the critic as ω, the update rule of the state-action critic is defined based on the MSE loss function (Equation 2.13) between the critic estimation and the target value:

$$\omega_{t+1} \leftarrow \omega_t - \beta_\omega \nabla_\omega \mathbb{E}^{\pi_\theta}_{s\sim\rho^{\pi_\theta}} \left[L\left(Q^{\pi_\theta}_{\omega_t}(s,a), Q^{\pi_\theta}_{target}(s,a) \right) \right] \tag{2.18}$$

The target value depends on the off-policy or on-policy variant of the RL algorithm. On-policy methods such as TRPO [8] and PPO [9], use the reward-to-go of the state-action as a Monte Carlo return of following the policy to reach the end of the episode. In the case of off-policy, the Bellman error is used as the updating target:

$$Q^{\pi_\theta}_{target}(s,a) = r(s,a) + \gamma Q^{\pi_\theta}_{\omega_t}(s',a') \tag{2.19}$$

where $s', r \sim \mathcal{P}(s, a)$ and $a' \sim \pi_\theta(s')$. It should be noted that while the target value can also be dependent on the parameters of ω, the gradient is not calculated for it. Therefore, the evaluation of the derivative is solely based on $Q^{\pi_\theta}_{\omega_t}(s, a)$.

While the policy gradient theorem is a powerful tool for formulating RL algorithms with stochastic policies, taking the integration over the action space will be infeasible in RL tasks with high-dimensional or continuous action spaces. To this end, the Deterministic Policy Gradient (DPG) Method [10] proposes the use of a deterministic policy. In this context, the stochastic policy creates a probability distribution as the output. On the other hand, a deterministic policy outputs a single action as the output. Hence, the objective function of the policy can be written as:

$$\mathcal{J}_\pi(\theta) = \int_{s \in \mathcal{S}} \rho^{\mu_\theta}(s) Q(s, \mu_\theta(s)) \mathbf{d}s \tag{2.20}$$

where the deterministic policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$. Hence, the gradient of the objective function can be calculated using the chain rule:

$$\nabla_\theta \mathcal{J}_\pi(\theta) = \int_{s \in \mathcal{S}} \rho^{\mu}(s) \nabla_a Q^{\mu}(s, a) \nabla_\theta \mu_\theta(s) \mathrm{d}s = \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_a Q^{\mu}(s, a) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)} \right] \tag{2.21}$$

Finally, based on the DPG formulation, Deep Deterministic Policy Gradient [11] algorithm uses neural networks as the parametrization of the actor and critic. Furthermore, the deterministic policy formulation suffers from the risk of not exploring the environment adequately. For that purpose, the authors proposed the addition of noise to the policy output:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N} \tag{2.22}$$

where $\mathcal{N}$ is an Ornstein-Uhlenbeck process [12].

2.2.3 Soft Actor-Critic

While DDPG offers a powerful baseline in continuous control tasks, it tends to be unstable in the training process and due to lack of enough exploration, it may converge to a suboptimal solution. In order to address these issues, the soft actor-critic (SAC) method [13] proposes an stochastic policy and updating it using the maximum entropy framework. Under this framework, the objective is to maximize the expected sum of discounted return and the policy entropy:

$$\mathcal{J}_\pi(\theta) = \mathbb{E}^{\pi_\theta}_{s_t \sim \rho^{\pi_\theta}} \left[\sum_{t=1}^{\infty} r_t + \alpha \mathcal{H}(\pi_\theta(s_t)) \right] \tag{2.23}$$

where $\mathcal{H}$ is the entropy measure of the policy and α is the weighing coefficient called temperature. Entropy is a measure of uncertainty. In the context of the RL policy, the higher the entropy of the policy is in a given state, the more uncertain it is about what the optimal action is. Therefore, the addition of the entropy term to the objective function rewards the policy for exploring states previously unseen states to decrease its uncertainty while maximizing the return. To this end, the Shannon entropy for a random variable X is defined as:

$$H(X) = \mathbb{E}[-\log p(X)] = -\int p(x) \log p(x) \tag{2.24}$$

Based on the SAC objective function in Equation 2.23, the Bellman equation for the soft Q-value value can be derived as:

$$Q(s,a) = r(s,a) + \gamma \mathbb{E}^{\pi_\theta}_{s' \sim \mathcal{P}(s,a)} \left[Q(s',a') - \alpha \log \pi(a'|s') \right] \tag{2.25}$$

Using the soft Bellman error, the critic can then be trained using a similar objective formulation as in Equation 2.18:

$$\mathcal{J}_Q(\omega) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q^{\omega}(s,a) - \mathbb{E}^{\pi_\theta}_{\mathcal{P}}\left[r(s,a) + \gamma Q(s',a') - \alpha \log \pi(a'|s')\right]\right)^2\right] \quad (2.26)$$

where $\mathcal{D}$ is called the replay buffer which stores the one-step transitions in the training process as a tuple (s, a, r, s').

While the training of the critic can be done in a similar fashion as in previous works in the literature, the training of the policy cannot be done using the previous formulations for continuous RL since the policy of SAC is a stochastic distribution over the action space. To this end, the authors proposed minimizing the KL divergence between the policy distribution and the $exp(Q^{\pi_\theta}(s,a))$:

$$\begin{aligned}
\mathcal{J}_\pi(\theta) &= D_{\mathrm{KL}}\left(\pi_\theta(.|s)||\frac{\exp\left(Q_\omega(s,.)\right)}{Z_\omega(s)}\right) \\
&= \mathbb{E}_{a\sim\pi}\left[-\log\left(\frac{\exp\left(Q_\omega(s,a) - \log Z_\omega(s)\right)}{\pi(a|s)}\right)\right] \\
&= \mathbb{E}_{a\sim\pi}\left[\log \pi_\theta(a|s) - Q_\omega(s,a) + \log Z_\omega(s)\right]
\end{aligned} \quad (2.27)$$

where Z is the partition function for normalizing the reference distribution. While the partition function is generally intractable, it can be eliminated since it does not contribute to the gradient. Furthermore, the calculation of $log\ \pi$ requires a tractable distribution function. For that reason, the authors use the Gaussian distribution as the policy distribution. Moreover, SAC uses a reparameterization trick to evaluate the objective function of SAC:

$$\mathbf{a} = f_\theta(s) = \mu_\theta(s) + \sigma_\theta^{\mathrm{T}}\mathcal{N}(0,1) \quad (2.28)$$

Therefore, the gradient of the policy objective can be calculated as:

$$\nabla_\theta \mathcal{J}_\pi(\theta) = \alpha \nabla_\theta \log \pi_\theta(\mathbf{a}|s) + (\alpha \nabla_{\mathbf{a}} \log \pi_\theta(a|s) - \nabla_{\mathbf{a}} Q(s, \mathbf{a}))\, \nabla_\theta f_\theta(s) \tag{2.29}$$

The policy parameters are then updated as:

$$\theta_{t+1} \leftarrow \theta_t - \beta_\theta \nabla_\theta \mathcal{J}_\pi(\theta_t) \tag{2.30}$$

While SAC provides an effective RL formulation for continuous action spaces with stochastic policy and intrinsic exploration formulation, the value of temperature α can impact the overall performance of the algorithm. Hence, α requires adequate hyperparameter tuning. For that reason, the second version of SAC [14], provided an automatic temperature tuning framework. To do so, the author reformulated the objective function as a constrained objective:

$$\max_{\pi_0, \ldots, \pi_T} \mathbb{E}_{\rho^\pi} \left[\sum_{t=0}^{T} r(s_t, a_t) \right] \text{ s.t. } \forall t, \mathcal{H}(\pi_t) \geq \mathcal{H}_0 \tag{2.31}$$

where $\mathcal{H}_0$ is a predefined policy entropy threshold. The authors start by deriving the optimal policy formulation at time the final time T. To do so the constrained objective maximization can be converted to the Lagrangian problem:

$$\begin{aligned} &\max_{\pi_T} \mathbb{E}_{\rho^\pi}[r(s_T, a_T)] \text{ s.t. } \mathbb{E}_{(s_T, a_T) \sim \rho^\pi}[-\log(\pi_T(a_T|s_T))] \geq \mathcal{H}_0 \\ &= \min_{\alpha_T \geq 0} \max_{\pi_T} \mathbb{E}_{\rho^\pi}[r(s_T, a_T) - \alpha_T \log \pi(a_T|s_T)] - \alpha_T \mathcal{H}_0 \end{aligned} \tag{2.32}$$

where α_T (called temperature in the first version of SAC) is the Lagrangian multiplier. Furthermore, solving the Lagrangian expression and going back recursively through time, the authors show that the objective function that can achieve the optimal value for α can be written as:

$$\mathcal{J}_\alpha(\alpha) = \mathbb{E}^{\pi}_{\rho^\pi}[-\alpha \log \pi(a|s) - \alpha\mathcal{H}_0] \tag{2.33}$$

The objective function for the gradient of the policy will be the same as in Equation 2.29. Finally, for a given task, the authors propose using the policy entropy threshold of $\mathcal{H}_0 = -dim(\mathcal{A})$.

2.3 Safe Reinforcement Learning

In the previous section, the main concern for the development of RL algorithms were to maximize the expected return in a specific environment. Hence, the methods were designed to explore the state space as much as possible to find the optimal policy. However, there is a class of RL problems where in addition to maximizing the return, it is important to adhere to certain constraints in the environment. The most important application of constraints in this context is the safe RL where the constraints are defined based on the physics of the system. The aim of safe RL algorithms is to design the training pipeline of the policy such that the constraint violations during the training process is minimized and the final converged policy performs safely.

2.3.1 Constrained Markov Decision Process

In the context of safe RL, the Constrained MDP (CMDP) is represented as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, c, \gamma, \gamma_c \rangle$ where in addition to the definitions in Section 2.1.1, the constraint signal $c : \mathcal{S} \rightarrow [0, 1]$ determines whether a specified state violates the constraint functions specified by C: $c(s) = \mathbb{1}[C(s) == 1]$, and γ_c specifies the discount factor of the constraint value function. To this end, the constraint state-action value function is formulated as:

$$Q_c^\pi(s,a) = \mathbb{E}_\mathcal{P}^\pi \left[c(s) + (1 - c(s)) \sum_{t=1}^{\infty} \gamma_c^t c(s_t) \right] \tag{2.34}$$

The idea of the definition of constraint value function is that it aims to estimate the probability of failure in the future by following the policy π. Hence, the lower the Q-value, the safer the policy will be. Furthermore, the Bellman equation for the constraint value function can be derived as:

$$Q_c^\pi(s,a) = \Pr[c(s) = 1] + \gamma_c \mathbb{E}_\mathcal{P}^\pi \left[(1 - c(s)) Q_c^\pi(s', a') \right] \tag{2.35}$$

Using the definition of the constraint state-action value function, the constraint objective function can be defined similar to the return objective:

$$\mathcal{J}_c(\theta) = \mathbb{E}_{s\sim\rho^\pi}^{\pi_\theta} \left[c(s) + (1 - c(s)) \sum_{t=1}^{\infty} \gamma_c^t c(s_t) \right] = \int_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \int_{a\in\mathcal{A}} \pi_\theta(a|s) Q_c^{\pi_\theta}(s,a) \mathrm{d}a\mathrm{d}s \tag{2.36}$$

Finally, it is important to note that the definition of the CMDP provided in this section is based on the state-wise CMDP formulation [15]. Under this framework, the constraints must be satisfied at all time otherwise the episode will be terminated. There are also other formulations of safe RL available involving soft constraints or budget-based constraint violation. However, in this research, the definition of state-wise safety is being focused on because it provides the most real-world applicable framework since most safety constraints defined in the context of real-world applications, such as keeping the vehicle and the passenger healthy at all times for autonomous driving, are hard constraints and guarantees based on this framework can be more useful.

2.3.2 Lagrangian-based Methods

As discussed in the previous section, the aim of a safe RL method is to maximize the expected return during the interaction process with the environment while also optimizing the safety performance during the training process. Hence, the problem is inherently a multi-objective optimization problem. However, one of the most common approaches to tackle this problem is through bounding safety objective and formulating the problem as constrained optimization:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \ \mathcal{J}_r(\pi) \text{ s.t. } \mathcal{J}_c(\pi) \leq \varepsilon \tag{2.37}$$

where ε is the constraint objective threshold that determines to what extent should the policy consider safety during its training process. To differentiate between the reward and the safety performance, $\mathcal{J}_r$ notation is used for reward performance which can be any of the objectives mentioned in Section 2.1 (such as the ones discussed in Equations 2.15, 2.20, 2.23). Furthermore, the constrained optimization problem can be converted to the Lagrangian loss to be solved:

$$\pi^*, \nu^* = \min_{\nu \geq 0} \ \max_{\pi \in \Pi} \ \mathcal{L}(\pi, \nu) = \mathcal{J}_r^{\pi}(\theta) - \nu(\mathcal{J}_c^{\pi}(\theta) - \varepsilon) \tag{2.38}$$

where ν is the Lagrangian multiplier. Various algorithms have been proposed to optimize this Lagrangian loss. In its basic form, the policy (parameterized as θ) and the Lagrangian multiplier are updated by taking the gradient w.r.t. the Lagrangian loss:

$$\begin{aligned} \theta_{t+1} &\leftarrow \theta_t + \beta_\theta \nabla_\theta \mathcal{L}(\pi_\theta, \nu) = \theta_t + \beta_\theta \left(\nabla_\theta \mathcal{J}_r^{\pi}(\theta) - \nu \nabla_\theta \mathcal{J}_c^{\pi}(\theta)\right) \\ \nu_{t+1} &\leftarrow \nu_t - \beta_\nu \nabla \mathcal{L}(\pi_\theta, \nu) = \nu_t + \beta_\nu (\mathcal{J}_c^{\pi}(\theta) - \varepsilon) \end{aligned} \tag{2.39}$$

This formulation and updating rule is called Lagrangian Relaxation (LR). The Lagrangian suffix may also be added depending on the definition of the policy reward objective (such as SAC-Lag or PPO-Lag).

There are other methods that aim to further enhance the performance of Lagrangian-based safe RL. Constrained Policy Optimization (CPO) [16] was the first approach policy gradient method for constrained RL which provided near-constraint satisfaction in each iteration of training. Risk Sensitive Policy Optimization (RSPO) [17] optimizes the Lagrangian loss with a decreasing sequence to 0. Reward Constrained Policy Optimization (RCPO) [18] modifies the reward function as $\hat{r}(s,a) = r(s,a) - \nu c(s')$ and proposes the value function formulation of:

$$\hat{Q}^{\pi}(s,a) = Q_r^{\pi}(s,a) - \nu Q_c^{\pi}(s,a) \tag{2.40}$$

Hence, instead of taking the gradient of the Lagrangian loss, the critic value function is used directly to optimize the policy. Interior-point Policy Optimization [19] proposed a first-order policy optimization based on interior-point optimization augmented with a logarithmic barrier function to satisfy safety. Projection-based Constrained Policy Optimisation (PCPO) [20], proposed a two-step method which first optimizes the policy toward better reward performance then projects the policy onto the safe set to guarantee safety. Finally, RC-D4PG [21] proposed optimizing the Lagrangian learning rate using meta-gradient optimization. [22, 23, 24, 15] provide more comprehensive review of constrained RL methods.

2.3.3 Safe Exploration Methods

Another approach in the literature to tackle the problem of Safe RL is through modifying unsafe actions locally, meaning they modify the action when it is identified as

leading to an unsafe state. In Safety Q-Function for RL (SQRL) [25], the authors proposed using rejection sampling to filter the actions proposed by the policy safety value of the action. [26] applies a modification to the action of the policy using a safety layer that analytically solves an action correction formulation to minimally change the action so that the safety constraints are satisfied. Furthermore, Safety Editor [27] trains a safety editing policy that maximizes the expected return while minimizing the hinge loss of the state-action value difference before and after editing. [28] introduce a learning-based model predictive control (MPC) framework that utilizes a reliable statistical model to guarantee accurate confidence interval on the output trajectories. [29] introduced an unsupervised action planning method that stores the agent's recovery actions for leaving unsafe areas in a dedicated replay buffer, subsequently utilizing it when the agent faces an unsafe state. Safe model-based policy optimization (SMBPO) [30] plans a brief horizon into the future to anticipate and prevents safety violations by applying penalties to unsafe trajectories. Finally, Recovery RL [31] uses a backup policy trained to ensure safety to recover the agent from unsafe regions caused by the main reward policy and it is executed whenever an action chosen by the main is deemed too risky by the safety critic (the value is $> \varepsilon_{risk}$).

2.4 Meta-Gradient Reinforcement Learning

The algorithms and formulations provided in the previous sections include many hyperparameters that their value can have meaningful effect on the performance of the RL method. For example, the choice of γ determines the degree of far-sightedness of how far into the future does the policy shape its optimal policy. Therefore, these meta-parameters (in this research the terms hyperparameter and meta-parameter terms are used interchangeably) dictate the dynamics of the system and direct it toward a

certain behavior. To produce a favorable performance, the RL practitioner typically needs to iteratively rollout a specific algorithm in the specific environment and update the hyperparameters of interest to examine the change in the performance. The process of rolling out and updating the hyperparameter continues until a favorable performance has been achieve. With the computational intensity of RL algorithms, tuning the value of the meta-parameters of interest can be rendered as a challenge since it requires many iterations of rollout. To address this challenge in the context of RL, one of the important method is proposed as the meta-gradient reinforcement learning [32] framework.

In abstract terms, consider the learnable system variables parameterized as θ. The parameters are updated to θ' by following a specific update rule:

$$\theta' \leftarrow \theta + f(\mathcal{J}, \theta, \eta, \mathcal{B}) \tag{2.41}$$

where η is a list of meta-parameters, $\mathcal{B}$ a batch of experience, and f is the gradient of the objective function J w.r.t. θ. Furthermore, based on a new batch of samples, the effect of the meta-parameter can be evaluated by taking the gradient of the meta-objective w.r.t. the meta-parameter by applying the chain rule:

$$\frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \eta} = \frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \theta'} \frac{\mathrm{d}\theta'}{\mathrm{d}\eta} \tag{2.42}$$

where $\mathcal{J}'$ is the meta-objective and $\mathcal{B}'$ is the new batch of samples. Basically, the calculation based on the meta-objective and the updated parameters θ' forms a differentiable function which allows for taking the gradient w.r.t. the meta-parameter. In order to evaluate Equation 2.42, the gradient of $\mathrm{d}\theta'/\mathrm{d}\eta$ can be computed using the update rule of θ (Equation 2.41):

$$\begin{aligned}\frac{\mathrm{d}\theta'}{\mathrm{d}\eta} &= \frac{\mathrm{d}\theta}{\mathrm{d}\eta} + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \eta} + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \theta}\frac{\mathrm{d}\theta}{\mathrm{d}\eta} \\ &= \left(I + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \theta}\right)\frac{\mathrm{d}\theta}{\mathrm{d}\eta} + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \eta}\end{aligned} \tag{2.43}$$

Hence, the update rule for the meta-parameter follows:

$$\eta' \leftarrow \eta + \beta_\eta \frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \eta} = \eta + \beta_\eta \frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \theta'}\frac{\mathrm{d}\theta'}{\mathrm{d}\eta} \tag{2.44}$$

Finally, it is important to note that the formulation of meta-gradient discussed in this research is based on the cross-validation method in the meta-optimization literature [33, 34] which is slightly different the formulation in [32], In the original paper, the authors propose rolling out the algorithm with the parameters θ' and use the data gathered from θ' as cross-validation for evaluating the effect of the meta-parameters. This formulation is suitable for on-policy algorithms. However, in the off-policy case, since it is possible to have access to the past experience, the cross-validation can be reduced to resampling the batch of experience.

Chapter 3

Meta Soft Actor-Critic Lagrangian

3.1 Introduction

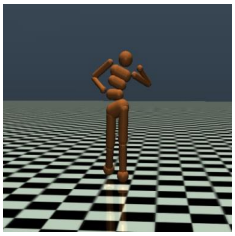
As discussed in Section 2.3, the most common approach in safe RL is through the Lagrangian method. Specified under the Constrained Markov Decision Process (CMDP) framework, through defining thresholds for the constraints, the multi-objective optimization problem is converted to constraint satisfaction and is solved by casting it to an unconstrained problem using the Lagrangian method. While the approach has been extensively studied in the literature [22, 24], without precise tuning and engineering of the constraint thresholds, the Lagrangian methods will suffer from convergence to suboptimal policies. For that reason, the real-world use of these algorithms is rendered to be challenging due to the iterative process of hyperparameter tuning. For instance, as discussed in [35], the value of policy entropy threshold can be effective on the overall performance of the method.

To address these challenges, based on the Soft Actor-Critic (SAC) architecture [13] (discussed in Section 2.2.3), the algorithm proposed in the chapter aims to address two fundamental problems: safe exploration and tuning-free constraint adjustment.

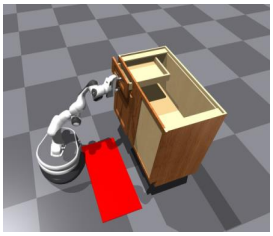
Previous attempts to automate the tuning of exploration-related hyperparameter of SAC have been mostly focused on optimizing the performance of the system [14, 36]. However, addressing the safety compliance of SAC has been limited. To this end, a threshold-free safety-aware exploration optimization pipeline called Meta Soft Actor-Critic Lagrangian (Meta SAC-Lag) is proposed. In addition, the proposed approach optimizes the safety threshold according to the overall performance of the policy. Finally, to assess the performance of Meta SAC-Lag, as depicted in Figure 3.1, its performance is studied against several baseline algorithms in five simulated robotic tasks with four different application themes. It is observed that the method attains better or comparable results in terms of safety or reward performance while automatically tuning the safety-related hyperparameters. Furthermore, a safety benchmark test case is proposed, called *Pour Coffee*, which attempts to relocate and pour a coffee-filled mug into another cup. Constraint violation happens in case of collision or the spillage of the coffee. Meta SAC-Lag is deployed and trained in a real-world setup using Kinova Gen3 robot. The implementation shows that not only is Meta SAC-Lag capable of safe deployment without the iterative process of hyperparameter tuning but also, the learning process of the policy results in a smooth and jerk-free execution of the task with minimum effort imposed on the system.

The main contributions in this chapter can be summarized as:

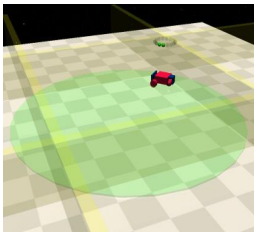
- A Lagrangian-based safe RL method able to automatically adjust the constraint bounds is proposed.
- Meta SAC-Lag addresses safe exploration through an unconstrained metagradient-based optimization pipeline.
- The applicability of Meta SAC-Lag is validated in five simulated robotic environments against baseline algorithms.



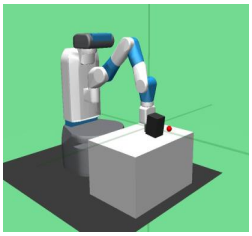
(a) Humanoid-Velocity



(b) Franka CloseDrawer

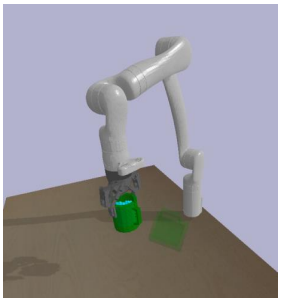


(c) Car-Circle2



(d) Fetch PushTopple

(e) Egg Manipulate



(f) Pour Coffee (simulation)

(g) Pour Coffee (real setup)

Figure 3.1: Safety-critical environments used to deploy Meta SAC-Lag. The top row represents simulated environments with four general safety topics: locomotion (a), obstacle avoidance (b,c), robotic manipulation (d), dexterous manipulation (e). The bottom row represents Pour Coffee environment (f,g) used to study the deployability of the algorithm in a real-world setup.

- A test environment, called *Pour Coffee*, is presented, and, with minimal prior safety-related hyperparameter tuning, Meta SAC-Lag is trained on a real-world Kinova Gen3 setup. The algorithm successfully achieves the task objective with minimized effort exerted on the robot.

3.2 Methodology

In this section the process of metagradient optimization of the safety threshold ε and entropy temperature α is discussed.

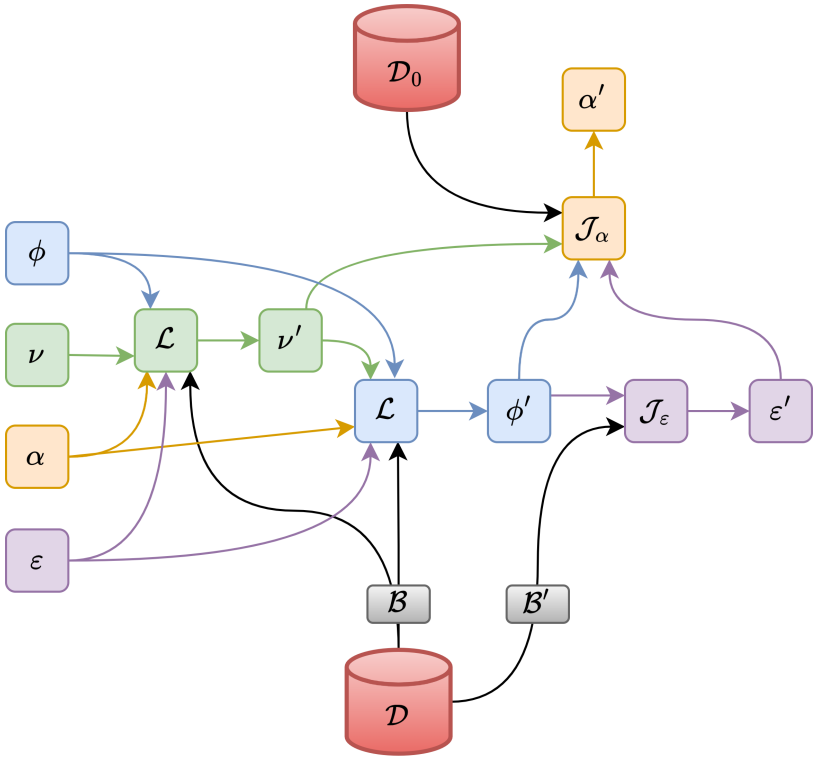


Figure 3.2: The Computational Graph of the Meta SAC-Lag.

3.2.1 SAC-Lagrangian

In the Lagrangian version of the SAC the aim is to optimize the policy based on its reward objective such that it is compliant with the safety objective:

$$\begin{aligned} \pi_\phi^* = \max_{\pi_\phi \in \Pi} \; \mathcal{J}_{r_{\pi_\phi}}^{\pi_\phi} &= \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi}} [Q_{\omega_r}(s,a) - \alpha \log \pi_\phi(a|s)] \\ \text{s.t.} \quad \mathcal{J}_c^{\pi_\phi} &= \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi}} [Q_{\omega_c}(s,a)] \leq \varepsilon \end{aligned} \tag{3.1}$$

Naturally, multiple constraints can be defined for the policy to consider all of them. However, in order to keep the formulation simple and general, a single constraint signal is considered that is the result of the superposition of all the constraint functions. In this research, in contrast to [13], α is not considered as an additional constraint and is instead optimized through metagradient optimization.

Furthermore, the optimization process of policy in Equation 3.1 is formulated by

casting it as a Lagrangian loss and backpropagating through the loss:

$$\begin{aligned}\min_{\nu\geq 0}\ \max_{\pi_\phi\in\Pi}\ \mathcal{L}(\pi_\phi,\nu,\varepsilon,\alpha) &= \mathcal{J}_{r_{\pi_\phi}}^{\pi_\phi} - \nu(\mathcal{J}_c^{\pi_\phi}-\varepsilon)\\ &= \mathbb{E}_{\substack{s\sim\mathcal{D}\\ a\sim\pi_\phi}}\left[Q_{\omega_r}(s,a) - \alpha\log\pi_\phi(a|s) - \nu(Q_{\omega_c}(s,a)-\varepsilon)\right]\end{aligned} \tag{3.2}$$

where ν is the Lagrange multiplier.

3.2.2 Meta SAC-Lag

Following the conventional notation in the context of gradient-based hyperparameter optimization [34], the parameters are split into *inner* and *outer* parameters. Rather than a one-shot optimization as in Equation 2.41, a sequential updating approach is proposed. Hence, the inner parameters are defined and updated as:

$$\theta'_{\text{inner}} = \begin{bmatrix}\nu'\\ \phi'\end{bmatrix} = \begin{bmatrix}\nu\\ \phi\end{bmatrix} + \begin{bmatrix}-\nabla_\nu\mathcal{L}(\pi_\phi,\nu,\varepsilon,\alpha)\\ \nabla_\phi\mathcal{L}(\pi_\phi,\nu',\varepsilon,\alpha)\end{bmatrix} \tag{3.3}$$

Furthermore, in the same sequential manner, ε is updated first and the α:

$$\theta'_{\text{outer}} = \begin{bmatrix}\varepsilon'\\ \alpha'\end{bmatrix} = \begin{bmatrix}\varepsilon\\ \alpha\end{bmatrix} + \begin{bmatrix}\nabla_\varepsilon\mathcal{J}_\varepsilon(\pi_{\phi'})\\ \nabla_\alpha\mathcal{J}_\alpha(\pi_{\phi'},\nu',\varepsilon')\end{bmatrix} \tag{3.4}$$

where $\mathcal{J}_\varepsilon$ and $\mathcal{J}_\alpha$ correspond to the objective functions of ε and α, respectively. To this end, the intent behind the design of the objective function for ε was solely based on the performance of the resultant policy. The intuition behind the aforementioned design stems from the idea that the threshold should be adjusted such that it improves the performance of the agent as a whole. For that purpose, the ε objective function is proposed as:

Algorithm 1 Meta SAC-Lag

Require:

Initialize Policy network ϕ^0, Exploration rate α^0
Critic network $\omega^0_{r_1}, \omega^0_{r_2}$, Safety critic network $\omega^0_{c_1}, \omega^0_{c_2}$
Lagrangian values ε^0, ν^0
Learning rates $\beta_\phi, \beta_\varepsilon, \beta_\nu, \beta_\alpha$

Create Transition buffer $\mathcal{D}$, Safety buffer $\mathcal{D}_\text{s}$, and Initial state buffer $\mathcal{D}_0$
Randomly sample initial state $s_0 \sim \rho_0$ and fill $\mathcal{D}_0$
for $e = 1, \ldots$ **do**
    Reset environment $s_0 \sim \rho_0 = env.reset()$
    for $t = 0, \ldots, T-1$ **do**
        Sample action $a_t \sim \pi_\phi$
        $s_{t+1}, r_t, c_t \leftarrow env.step(a_t)$
        if $c_t == 1$ **then**
            $\mathcal{D}_c \leftarrow \mathcal{D}_c \cup (s_t, a_t, c_t, s_{t+1})$
        else
            $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, r_t, c_t, s_{t+1})$
        Train $\omega_{c_1}, \omega_{c_2}$ on $\mathcal{D} \cup \mathcal{D}_c$ (Eq. 2.35)
        Sample a batch of transitions $\mathcal{B} = \{(s, a, r, c, s')\} \in \mathcal{D}$
        Train $\omega_{r_1}, \omega_{r_2}$ using $\mathcal{B}$ (Eq. 2.26)
        $\nu' \leftarrow \nu - \beta_\nu \nabla_\nu \mathcal{L}(\pi_\phi, \nu, \varepsilon, \alpha)$ using $\mathcal{B}$ (Equation 3.2)
        $\phi' \leftarrow \phi + \beta_\phi \nabla_\phi \mathcal{L}(\pi_\phi, \nu', \varepsilon, \alpha)$ using $\mathcal{B}$ (Equation 3.2)
        Resample $\mathcal{B}' = \{s \in \mathcal{D}\}$
        $\varepsilon' \leftarrow \varepsilon + \beta_\varepsilon \nabla_\varepsilon \mathcal{J}_\varepsilon(\pi_{\phi'})$ using $\mathcal{B}'$ (Eq. 3.5)
        $\alpha' \leftarrow \alpha + \beta_\alpha \nabla_\alpha \mathcal{J}_\alpha(\pi_{\phi'}, \nu', \varepsilon')$ using $\mathcal{D}_0$ (Eq. 3.6)
        $\nu \leftarrow \nu', \phi \leftarrow \phi', \varepsilon \leftarrow \varepsilon', \alpha \leftarrow \alpha'$
        if $c_t == 1$ **then** Break

$$\mathcal{J}_\varepsilon(\pi_{\phi'}) = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{\phi'}}} [\nu'_\text{copy} Q_{\omega_c}(s, a) - Q_{\omega_r}(s, a)] \tag{3.5}$$

The objective function $\mathcal{J}_\varepsilon$ is consistent with the objective function of the policy. This is evident by comparing Equation 3.5 with the gradient w.r.t. the policy parameters ϕ in Equation 3.2. The objective function for ε is designed to minimize the policy objective. This design stems from the idea that ε aims to capture the worst-case performance of the policy $\pi_{\phi'}$. Hence, by being optimized in this way, the safety region of the policy can be correctly adjusted. The reasoning behind the

choice of $\mathcal{J}_\varepsilon$ is further discussed in Section 3.3.3. It is important to note that ν'_{copy} is used to indicate that ν' value is merely used in the objective function and is not included its gradient w.r.t. ε. Better performance by the gradient detachment in our the experiments was observed which may be due to the injection of bias in ν' into its optimization process. Furthermore, to optimize the exploration value α, [36] used Q_{ω_r} as the objective function to change the value based on the performance of the policy. Therefore, in order to make the exploration rate of the Meta SAC-Lag safety compliant, the objective function of α is proposed as:

$$\mathcal{J}_\alpha(\pi_{\phi'}, \nu', \varepsilon') = \max_{0<\alpha\leq 1} \mathbb{E}_{\substack{s_0\sim\rho_0 \\ a\sim\pi^{\text{det}}_{\phi'}}} [Q_{\omega_r}(s_0, a) - \nu'(Q_{\omega_c}(s_0, a) - \varepsilon')] \tag{3.6}$$

where $\pi^{\text{det}}_{\phi'}$ indicates the deterministic action value output by the policy. Basically, the expectation of the Lagrangian formulation evaluated in the initial states encountered by the agent is used. To gain a better understanding of the gradient relations, illustration of the optimization process of Meta SAC-Lag is depicted in Figure 3.2.

3.2.3 Implementation Details

The learning process of Meta SAC-Lag is presented in Algorithm 1. The proposed algorithm utilizes three replay buffers for training. The main replay buffer $\mathcal{D}$ stores all the transitions occurred while interacting with the environment, safety replay buffer $\mathcal{D}_{\text{s}}$ stores all the transitions that have led to a constraint violation, and $\mathcal{D}_0$ builds an approximation of ρ_0 by generating samples from the distribution. A sampled batch $\mathcal{B} \subset \mathcal{D}$ is used to train the critic networks and the *inner* parameters ν and ϕ. Following that, as discussed in Section 2.4, a resampled $\mathcal{B}' \subset \mathcal{D}$ and $\mathcal{D}_0$ are used to train the meta-parameters ε and α, respectively. The resampling process is analogous to the meta-testing process and is used to reduce bias in the training of the outer

parameters [33, 34]. Moreover, following the original architecture [13], Meta SAC-Lag uses two copies of the critic and safety critic parameter networks to prevent the overestimation of the value functions. To this end, following the clipped double-Q method [37], the target values in Equation 2.19 are calculated:

$$\begin{aligned} Q_r^{target} &\leftarrow \min\{Q_{\bar{\omega}_{r_1}}, Q_{\bar{\omega}_{r_2}}\} \\ Q_c^{target} &\leftarrow \max\{Q_{\bar{\omega}_{c_1}}, Q_{\bar{\omega}_{c_2}}\} \end{aligned} \tag{3.7}$$

The $\bar{\omega}$ notation is used to indicate the target networks which are copies of the main networks updated with a time delay. Proposed in [11], the target networks aim to increase the stability of the training process and are calculated using the polyak averaging:

$$\bar{\omega} \leftarrow \tau\omega + (1-\tau)\bar{\omega} \tag{3.8}$$

The hyperparameter $\tau \in (0, 1)$ typically has a value near zero.

Finally, it is also worth mentioning, in contrast to the original SAC, RMSProp [38] is used instead of Adam to calculate the higher-order gradients of the parameters ν, ϕ, and ε since backpropagating through RMSProp seems to be more numerically stable [36].

3.3 Experimental Results

In this section, the performance of Meta SAC-Lag is evaluated. Specifically, the aim is to study two questions:

- How much does the added autonomy affect the performance of the algorithm compared to the baseline methods?
- How capable is Meta SAC-Lag to learn optimal performance in a real-world

setup while avoiding actions that might catastrophically damage the system?

3.3.1 Test Benchmarks and Baselines

In order to study how the proposed algorithm will perform in safety-critical robotic scenarios, five simulated robotic environments with four different themes:

- **Locomotion**: In this theme, the purpose of control is to move the robotic system in the forward direction. The safety constraints are violated whenever the actions of the controller make the system exceed its limits, e.g., the velocity is higher than a certain threshold or the robot is falling to the ground. For that purpose, the Mujoco-based [39] Humanoid-Velocity environment from the Safety Gymnasium codebase [40] is used. It is important to note that the safety-related reward shaping of this environment is removed to have a better understanding of the safety performance of the algorithms.

- **Obstacle Avoidance**: In many real-world robotic applications, there are mobile robots with manipulation capabilities. An important constraint of these systems is achieving their goal while avoiding certain regions in their surroundings. The Isaac Gym-based FreightFrankaCloseDrawer [41] is adopted. In this setup, the robot attempts to get near a drawer and close it while avoiding a red region. In addition, Car-Circle2 task [40] is used where the objective is to steer a car in a circular motion while avoiding collision with two walls.

- **Manipulation**: Another important area of safety-concerned robotic applications is manipulation. For that purpose, two embodied scenarios are used. For the **robotic manipulation** task Push Topple [42, 29] environment is adopted where the robotic arm must relocate a box without toppling it. Furthermore, in the **dexterous manipulation** scenario, the Egg Manipulate task where the

agent must rotate an egg to a specific orientation without dropping it or exerting a force of more than 20 N is used. For both tasks, the Gymnasium Robotics codebase [43] is utilized.

To provide concise information on the details of the environment, their state and action space dimensions and task descriptions are included in Table 3.1. It is important to note that in the training process, the constraints are treated as hard constraints and terminate the episode whenever a violation has happened in the system.

Table 3.1: Information about the environments used.

Task name	**dim($\mathcal{S}$)**	**dim($\mathcal{A}$)**	**Description**
Humanoid-Velocity [44]	376	17	Control a 3D bipedal humanoid move without exceeding a certain velocity and falling over to the ground.
Franka DrawerClose [40]	57	12	Control a mobile manipulator FrankFreight robot to get close to a drawer and close it without going over a forbidden red region.
Car-Circle2 [16, 45, 40]	40	2	Control a car to move on a circular path without colliding with two walls.
Fetch PushTopple [46]	28	4	Control a 7 DOF Fetch robot to push a box to a goal position without toppling it over.
Egg Manipulate [47, 46]	153	20	Control a Shadow Dexterous Hand to manipulate an Egg to reach a goal orientation without exerting force more than 20N.

Furthermore, three baseline algorithms are chosen to compare and study the performance of Meta SAC-Lag:

- **SACv2-Lag**: The basic form of Meta SAC-Lag which uses Equation 3.2 to optimize the policy and the Lagrangian multiplier with a fixed safety threshold.

- **Reward Constrained Policy Optimization (RCPO-SACv2)**: Optimizes

the policy using the Q-function formulated as $\mathbb{E}_{\pi}[\hat{Q}(s,a) = Q_r(s,a) - \nu Q_c(s,a)]$. The dual variable ν is also updated using Equation 3.2.

- **RCPO-MetaSAC**: To show the effectiveness of the safe exploration technique, the $\hat{Q}(s,a)$ formulation in RCPO (Equation 2.40) is used and α is optimized by using $\hat{Q}(s,a)$ as meta-objective.

- **Meta SAC-Lag** $\mathcal{J}_{nl}$: Inspired by [48], a nonlinear objective function for ε is experimented with. It is specified as:

$$\mathcal{J}_{\varepsilon}^{nl}(\pi_{\phi'}) = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{\phi'}}} \left[\begin{cases} Q_{\omega_r}(s,a)Q_{\omega_c}(s,a) & \text{if } Q_{\omega_r}(s,a) < 0 \\ Q_{\omega_r}(s,a)(1 - Q_{\omega_c}(s,a)) & \text{otherwise} \end{cases} \right] \tag{3.9}$$

  Essentially, $\mathcal{J}_{\varepsilon}^{nl}$ can have the advantage of no reliance on external parameter values, as opposed to Eq. 3.5 which uses ν' in the objective formulation.

To have a fair comparison, the values of ε and ν for SACv2-Lag and RCPO-SACv2 are tuned. Also, the values of RCPO-SACv2 are used for RCPO-MetaSAC. The values are outlined in Table 3.2. Furthermore, two important initial hyperparameter values of Meta SAC-Lag are automatically tuned; therefore, $\varepsilon = 1$ and $\alpha = 1$ are set as their initial values. Moreover, due to their similar training pipelines, the initial values of ν for SACv2-Lag in Table 3.2 are used for Meta SAC-Lag. Finally, the values of $\gamma_r = 0.99$ and $\gamma_c = 0.6$ are used for all the tasks. The results indicate the mean and variance of the performance of the algorithms across multiple independent runs.

Table 3.2: Hyperparameter values (ε and ν) of the comparison methods

Environment / Parameter	ε	Meta SAC-Lag SACv2-Lag	RCPO-SACv2 RCPO-MetaSAC
Humanoid-Velocity	0.4	10	10
Franka DrawerClose	0.6	10	10
Car-Circle2	0.5	100	1
Fetch PushTopple	0.5	1000	10
Egg Manipulate	0.5	100	1

3.3.2 Simulation Results

The simulation results are depicted in Figure 3.3. To this end, the final episodic return of rolling out the policy and the policy episodic violation rate are reported in Table 3.3. The violation rate is calculated as the average number of failures over a specific window of episodes. The results not only indicate that Meta SAC-Lag provides automated tuning of the safety-related hyperparameters but also, that the convergence process of the policy incurs lower constraint violations and yields higher or comparable returns. Furthermore, the update profile of α shows that as training goes on, in most cases, Meta SAC-Lag updates α to values lower than SACv2. This indicates that as the policy converges to a near-safe optimal solution, α is rapidly decreased to favor exploitation and prevent further constraint violations. Moreover, we can observe similar α profiles in Meta SAC-Lag and RCPO-SACv2 which can be attributed to α being optimized using similar objective functions. In addition, the optimization process of ε shows a generally fast convergence. The fast convergence of ε provides the advantage of stable optimization as other values can updated based

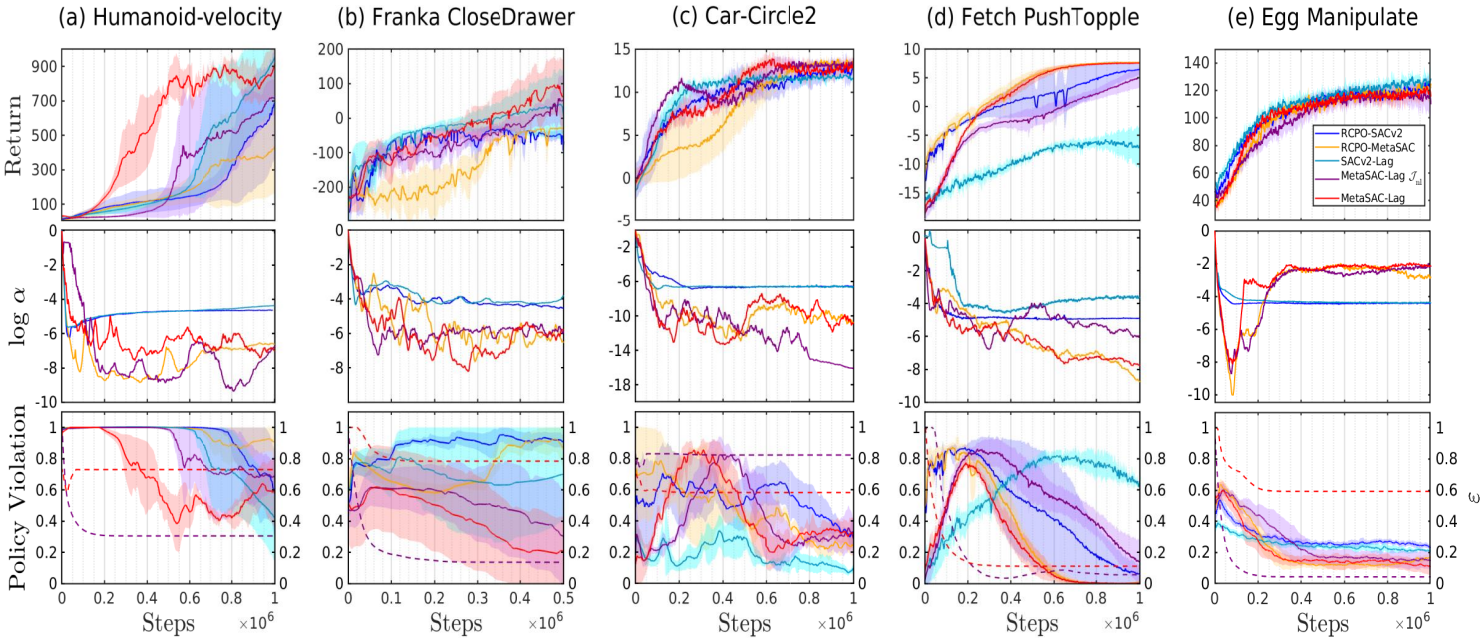


Figure 3.3: Performance of Meta SAC-Lag compared with the baseline algorithms. **(Top row):** Reward performance during the learning process. (Higher values are better) **(Middle row):** The value of Exploration hyperparameter (α). **(Bottom row):** Episodic policy safety performance of the algorithms during the learning process. (Lower values are better). The dashed lines illustrate the constraint threshold value (ε).

on the optimally achieved value of ε. Finally, regarding the comparison between Equation 3.5 and Equation 3.9 consistently better performance of Equation 3.5 is observed in both aspects of return and safety. In summary, the optimization outcomes of Meta SAC-Lag demonstrate that the algorithm excels across a range of embodied control tasks, proficiently learning optimal solutions, while demanding minimal hyperparameter tuning.

Table 3.3: Performance of the Algorithms During the Training Process
(The best performance is shown in bold)

Task / Method	SACv2-Lag		RCPO-SACv2		RCPO-MetaSAC		MetaSAC-Lag		MetaSAC-Lag $\mathcal{J}_{\text{nl}}$	
	Jr	Jc	Jr	Jc	Jr	Jc	Jr	Jc	Jr	Jc
Humanoid-Velocity	**956.11**	**0.42**	690.66	0.57	435.93	0.91	812.90	0.49	712.72	0.67
Franka DrawerClose	2.19	0.69	-67.20	0.89	-29.85	0.85	**63.13**	**0.20**	54.02	0.30
Car-Circle2	11.65	**0.10**	12.67	0.32	12.36	0.24	13.79	0.31	**13.95**	0.29
Fetch-Topple	-6.6	0.66	6.55	0.06	**7.6**	0.007	7.49	**0.004**	4.98	0.17
Egg Manipulate	**128.66**	0.21	121.53	0.23	124.99	0.17	115.44	**0.11**	109.95	0.14

3.3.3 Theoretical Analysis

In this section, the gradients of the each component of the algorithm is derived. While the automated differentiation tools for deep learning such as PyTorch [49] and TensorFlow [50] provide automated gradient calculation of this algorithm, the gradient analysis of Meta SAC-Lag may provide insightful information.

Step 1: In the beginning of optimization, the gradient of the Lagrangian multiplier ν is calculated using the inner loss (for ease of presentation, the expected values and the source of s are dropped in this analysis):

$$\begin{aligned}\nabla_\nu \mathcal{J}_\nu &= \nabla_\nu \mathcal{L}(\pi_\phi, \nu, \varepsilon, \alpha) \\ &= \nabla_\nu \left[Q_{\omega_r}(s, \pi_\phi(s)) - \nu(Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon) - \alpha \log \pi_\phi(a|s) \right] \\ &= -[Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon]\end{aligned} \tag{3.10}$$

Hence, ν is updated as:

$$\nu' \leftarrow \nu - \beta_\nu \nabla_\nu \mathcal{J}_\nu = \nu + \beta_\nu \left[Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon \right] \tag{3.11}$$

Step 2: Following the Lagrangian multiplier, the gradient of the actor parameters w.r.t. the Lagrangian loss is calculated as:

$$\begin{aligned}
&\nabla_\phi \mathcal{J}_\phi = \mathcal{L}(\pi_\phi, \nu', \varepsilon, \alpha) \\
&= \nabla_\phi \left[Q_{\omega_r}(s, \pi_\phi(s)) - \nu'(Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon) - \alpha \log \pi_\phi(a|s) \right] \\
&= \nabla_\phi \left[Q_{\omega_r}(s, \pi_\phi(s)) - (\nu + \beta_\nu (Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon)(Q_{\omega_c}(s, \pi_\phi(s)) - \varepsilon) - \alpha \log \pi_\phi(a|s) \right] \\
&= \nabla_\phi Q_{\omega_r}(s, \pi_\phi(s)) - \nabla_\phi Q_{\omega_c}(s, \pi_\phi(s)) \left[2\beta_\nu Q_{\omega_c}(s, \pi_\phi(s)) - 2\beta_\nu \varepsilon + \nu \right] - \alpha \nabla_\phi \log \pi_\phi(a|s)
\end{aligned} \tag{3.12}$$

The actor parameters are then updated as:

$$\begin{aligned}
&\phi' \leftarrow \phi + \beta_\phi \nabla_\phi \mathcal{J}_\phi \\
&\quad = \phi + \beta_\phi \big[\nabla_\phi Q_{\omega_r}(s, \pi_\phi(s)) - \nabla_\phi Q_{\omega_c}(s, \pi_\phi(s)) \left[2\beta_\nu Q_{\omega_c}(s, \pi_\phi(s)) - 2\beta_\nu \varepsilon + \nu \right] \\
&\qquad\qquad - \alpha \nabla_\phi \log \pi_\phi(a|s) \big]
\end{aligned} \tag{3.13}$$

Step 3: The first meta-parameter used in the training pipeline is the safety threshold ε. Assuming the meta-objective formulation of Equation 3.5, the gradient of ε can be derived as:

$$\begin{aligned}
\nabla_\varepsilon \mathcal{J}_\varepsilon &= \nabla_\varepsilon \left[\nu'_{\text{copy}} Q_{\omega_c}(s, \pi_{\phi'}(s)) - Q_{\omega_r}(s, \pi_{\phi'}(s)) \right] \\
&= \nabla_\varepsilon \phi'^{\mathbf{T}} \nabla_{\phi'} \mathcal{J}_\varepsilon + \nabla_\varepsilon \nu' \Big[\nabla_{\nu'} \phi'^{\mathbf{T}} \nabla_{\phi'} \mathcal{J}_\varepsilon + \cancelto{0}{\nabla_{\nu'} \mathcal{J}_\varepsilon} \Big] + \cancelto{0}{\nabla_\varepsilon \mathcal{J}_\varepsilon}
\end{aligned} \tag{3.14}$$

It should be noted that the chain of gradients that are calculated must be causal. For that reason, gradients such as $\nabla_{\phi'} \nu'$ would be meaningless and have not been written. Moreover, each component of Equation 3.14 can be computed as:

$$\nabla_{\varepsilon}\phi' = 2\beta_{\phi}\beta_{\nu}\nabla_{\phi}Q_{\omega_c}(s, \pi_{\phi}(s)) \tag{3.15}$$

$$\nabla_{\phi'}\mathcal{J}_{\varepsilon} = \nu'\nabla_{\phi'}Q_{\omega_c}(s, \pi_{\phi'}(s)) - \nabla_{\phi'}Q_{\omega_r}(s, \pi_{\phi'}(s)) \tag{3.16}$$

$$\nabla_{\varepsilon}\nu' = -\beta_{\nu} \tag{3.17}$$

$$\nabla_{\nu'}\phi' = -\beta_{\phi}\nabla_{\phi}Q_{\omega_c}(s, \pi_{\phi}(s)) \tag{3.18}$$

Therefore, the final gradient is calculated as:

$$\nabla_{\varepsilon}\mathcal{J}_{\varepsilon} = [-3\beta_{\nu}\beta_{\phi}\nabla_{\phi}Q_{\omega_c}(s, \pi_{\phi}(s))]^{\mathbf{T}} [\nabla_{\phi'}Q_{\omega_r}(s, \pi_{\phi'}(s)) - \nu'\nabla_{\phi'}Q_{\omega_c}(s, \pi_{\phi'}(s))] \tag{3.19}$$

Hence, the value of ε is updated as:

$$\begin{aligned}\varepsilon' &\leftarrow \varepsilon + \beta_{\varepsilon}\nabla_{\varepsilon}\mathcal{J}_{\varepsilon} \\ &= \varepsilon + \beta_{\varepsilon}[-3\beta_{\nu}\beta_{\phi}\nabla_{\phi}Q_{\omega_c}(s, \pi_{\phi}(s))]^{\mathbf{T}} [\nabla_{\phi'}Q_{\omega_r}(s, \pi_{\phi'}(s)) - \nu'\nabla_{\phi'}Q_{\omega_c}(s, \pi_{\phi'}(s))]\end{aligned} \tag{3.20}$$

The meta-gradient of ε consists of two components. While the right hand side evaluates the performance of the new actor parameter ϕ' by calculating the tradeoff between safety and optimality, the left hand side is the main direction driver of the gradient. The left hand side of the meta-gradient equation determines the policy's degree of unsafety with respect to the safety critic. The gradient of the policy w.r.t. the safety critic would determine the effect of the changes in the policy parameters on the measure of safety. The gradient will always try to force the policy to become safer by moving in the negative direction of the gradient. The idea of minimizing the policy objective discussed in Section 3.2.2 is also clear from the observation of the gradients of ε and specifically in Equation 3.16 and the left hand side of Equation 3.19.

Without the negative sign in the left hand side of the gradient, ε would change in the direction of increasing unsafety in the model.

Step 4: The final step of the optimization involves calculating the meta-gradient of the temperature α which is calculated as:

$$\begin{aligned}\nabla_\alpha \mathcal{J}_\alpha &= \nabla_\alpha [Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu'(Q_{\omega_c}(s, \pi^{det}_{\phi'}(s)) - \varepsilon')] \\ &= \cancelto{0}{\nabla_\alpha \nu'}[\nabla_{\nu'}\mathcal{J}_\alpha + \nabla_{\nu'}\phi'^{\mathbf{T}}\nabla_{\phi'}\mathcal{J}_\alpha + \nabla_{\nu'}\varepsilon'\nabla_{\varepsilon'}\mathcal{J}_\alpha] \\ &\quad + \nabla_\alpha \phi'^{\mathbf{T}}[\nabla_{\phi'}\mathcal{J}_\alpha + \nabla_{\phi'}\varepsilon'\nabla_{\varepsilon'}\mathcal{J}_\alpha] + \cancelto{0}{\nabla_\alpha \varepsilon'}\nabla_{\varepsilon'}\mathcal{J}_\alpha\end{aligned} \tag{3.21}$$

The components of the gradient can then be calculated as:

$$\nabla_{\varepsilon'}\mathcal{J}_\alpha = \nu' \tag{3.22}$$

$$\nabla_\alpha \phi' = -\beta_\phi \nabla_\phi \log \pi_\phi(a|s) \tag{3.23}$$

$$\nabla_{\phi'}\mathcal{J}_\alpha = \nabla_{\phi'} Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu' \nabla_{\phi'} Q_{\omega_c}(s, \pi^{det}_{\phi'}(s)) \tag{3.24}$$

$$\nabla_{\phi'}\varepsilon' = \beta_\varepsilon[-3\beta_\nu\beta_\phi \nabla_\phi Q_{\omega_c}(s, \pi_\phi(s))]^{\mathbf{T}} [H_{\phi'} Q_{\omega_r}(s, \pi_{\phi'}(s)) - \nu' H_{\phi'} Q_{\omega_c}(s, \pi_{\phi'}(s))] \tag{3.25}$$

Therefore, the final gradient of α is obtained as:

$$\begin{aligned}\nabla_\alpha \mathcal{J}_\alpha &= -\beta_\phi \nabla_\phi \log \pi_\phi(a|s)^{\mathbf{T}}[\nabla_{\phi'} Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu' \nabla_{\phi'} Q_{\omega_c}(s, \pi^{det}_{\phi'}(s)) \\ &+ \nu' \beta_\varepsilon[-3\beta_\nu\beta_\phi \nabla_\phi Q_{\omega_c}(s, \pi_\phi(s))]^{\mathbf{T}} [H_{\phi'} Q_{\omega_r}(s, \pi_{\phi'}(s)) - \nu' H_{\phi'} Q_{\omega_c}(s, \pi_{\phi'}(s))]] \\ &= -\beta_\phi \nabla_\phi \log \pi_\phi(a|s)^{\mathbf{T}}[\nabla_{\phi'} Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu' \nabla_{\phi'} Q_{\omega_c}(s, \pi^{det}_{\phi'}(s))]\end{aligned} \tag{3.26}$$

The second term of the gradient ($\nabla_{\phi'}\varepsilon'\nabla_{\varepsilon'}\mathcal{J}_\alpha$) was dropped because in the architecture of the neural networks used to implement Meta SAC-Lag, the ReLU activation function (Equation 2.11) was used. An important feature of ReLU is the fact that

it has zero second-order derivative almost everywhere. Hence, the Hessian matrix in Equation 3.25 would be equal to zero.

Finally, the update rule for ε follows:

$$\begin{aligned}\alpha' &\leftarrow \alpha + \beta_\alpha \nabla_\alpha \mathcal{J}_\alpha \\ &= \alpha - \beta_\alpha \beta_\phi \nabla_\phi \log \pi_\phi(a|s)^{\mathbf{T}} [\nabla_{\phi'} Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu' \nabla_{\phi'} Q_{\omega_c}(s, \pi^{det}_{\phi'}(s))]\end{aligned} \tag{3.27}$$

By observing the right hand side of $\nabla_\alpha \mathcal{J}_\alpha$ it can be noticed that term is equivalent to $\nabla_{\phi'}[Q_{\omega_r}(s, \pi^{det}_{\phi'}(s)) - \nu' Q_{\omega_c}(s, \pi^{det}_{\phi'}(s))]$ which is similar to the critic formulation of RCPO as $\hat{Q}^\pi(s,a) = Q^\pi(s,a) - \nu Q^\pi_c(s,a)$ in Equation 2.40. Hence, in essence the meta-gradient calculation of α in Meta SAC-Lag and RCPO-MetaSAC are the same. This observation can be confirmed by noticing the similarity in the updating profile of α in the simulation results in Figure 3.3. The profile is also similar to that of Meta SAC-Lag$\mathcal{J}_{nl}$ because due to the use of ReLU and the Hessian matrix becoming zero, the effect of $\mathcal{J}_\varepsilon$ on α has been eliminated and the updating procedure for both of them would be the same.

3.3.4 Real-World Deployment

Deployability can be regarded as one of the most important obstacles in using RL for learning to control real-world systems [51]. Choosing unsafe actions might lead the system to states that might damage it catastrophically, if chosen repeatedly. Therefore, using the conventional safe RL algorithms hinders their deployability since they require intensive hyperparameter tuning. In line with the purpose of assessing the deployability of a safe RL method, a simple, yet important, safe RL testbench is proposed. This task, which is called *Pour Coffee*, is the task of moving a coffee-filled mug from a home position to a specific location and pouring the coffee into

another cup. The task is executed using a Kinova Gen3 robot and its digital twin is created in the PyBullet simulation environment [52]. The state space is defined as $\mathcal{S} = \left\{ X_{cup} \cup O_{cup} \cup \dot{X}_{cup} \cup X_{goal} \cup O_{goal} \right\}$ where $X = \{x, y, z\}$ and $O = \{\psi, \theta, \phi\}$ refer to the Cartesian position and the Euler angles in the Tait-Bryan ZYX intrinsic convention, respectively. Furthermore, the action of the agent maps to the velocity of the end-effector: $\mathcal{A} = \{\dot{x}_{cup}, \dot{y}_{cup}, \dot{z}_{cup}, \dot{\phi}_{cup}\}$. Moreover, the reward function for reaching and pouring the coffee is defined hierarchically based on the Euclidean distance between the cup and the goal $d = ||X_{cup} - X_{goal}||_2$:

$$r(s, a, s') = \begin{cases} r_1 \cdot d + r_2 \cdot ||\ddot{X}_{cup}|| + r_3 \cdot \mathbb{1}[\text{spillage}] & \text{if } d > d_{\text{thresh}} \\ -|\phi_{cup} - \phi_{goal}| + 10 & \text{otherwise} \end{cases} \tag{3.28}$$

where $r_1 = -2$, $r_2 = -0.05$, $r_3 = -1$ and $d_{\text{thresh}} = 5\ cm$. Furthermore, the system violates the safety constraints whenever self-collision or collision with the environment objects occurs. Additionally, another constraint is defined as spilling the coffee. As will be shown, this constraint forces the policy to be less jerky and aims to minimize the acceleration. The advantage of this approach, in contrast to similar environments [53], is the fact that it will eliminate the need to explicitly engineer the reward function to minimize the jerk and acceleration of the robot.

Table 3.4: *Pour Coffee* Reward-Constraint Settings

Experiment Setting	Reward			Violation	
	Distance (r_1)	Acceleration (r_2)	Penalty (r_3)	Collision	Spillage
Simulation #1	✓	✗	✗	✓	✓
Simulation #2	✓	✓	✗	✓	✗
Simulation #3	✓	✓	✗	✓	✓
Simulation #4	✓	✓	✓	✓	✓
Real	✓	✗	✗	✓	✓

Meta SAC-Lag is experimented with in four reward and constraint settings. The experiments aim to study whether formulating the problem sub-objectives can be more practical by defining them as constraints rather than shaping the reward explicitly. In the presented task, coffee spillage provides an implicit sub-objective that can be explicitly modeled as the sub-objective of minimizing the jerk and acceleration of the end-effector during the execution of the task. As shown in Table 3.4, three experiments (Simulation #2, #3, #4) utilize different reward shaping schemes along with various constraint definitions. Moreover, the proposed method is trained without engineered reward shaping (Simulation #1) both in the simulation environment and the real-world Kinova Gen3 setup. In order to make comparisons and evaluate the Sim2Real capability, the simulation-trained models were deployed on the robot using the checkpoints saved during the learning process. The evaluation results are depicted in Figure 3.4. The results illustrate that, as a result of providing a denser reward signal, explicit reward shaping can have positive effects in the increase of the success rate. However, using the spillage constraint helps the algorithm be even more effort-compliant resulting in lower jerk and comparable acceleration results. In other words, while being successful in executing the task is the most important metric, in a

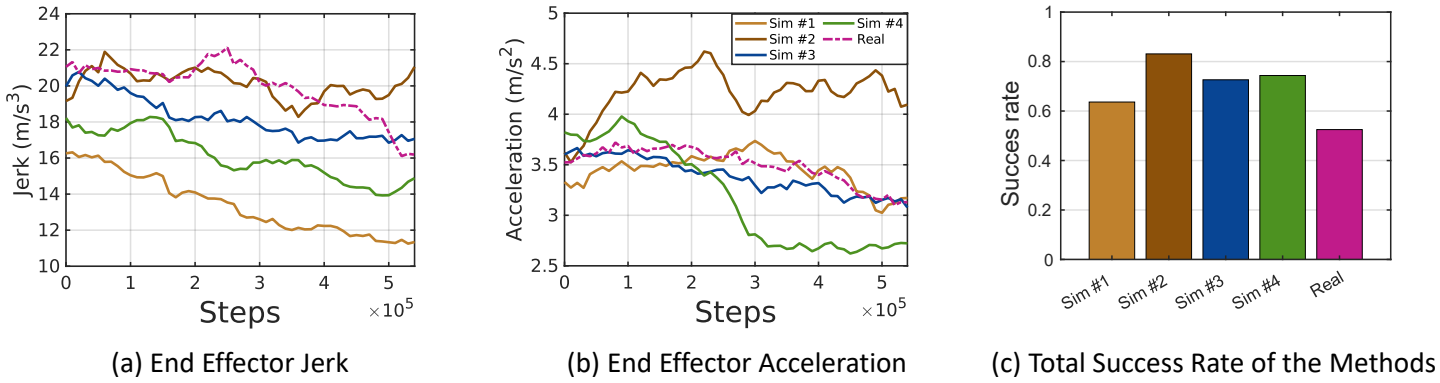


Figure 3.4: Deployment results of Meta SAC-Lag on the real-world setup. (a) and (b) represent the jerk and acceleration of the end effector during the training process. (c) shows the final success rate of the algorithms.

real-world scenario, sacrificing the performance to lower the effort of the system and satisfy other safety concerns can be reasonable. In addition, regarding the comparison between Sim2Real and Real deployment of the proposed algorithm, we can observe that while both setups have similar behaviors, the real-world deployment is slightly hindered by the system's physical limitations, such as sensor noise, control saturation, system fatigue, etc. Despite all that, the algorithm trained on the real-world setup without engineered reward function achieves results comparable to the models trained in the simulation.

3.4 Conclusions

This chapter focused on the problem of automatic hyperparameter tuning in Lagrangian safe RL methods. A novel model-free architecture called Meta SAC-Lag was proposed which addressed two inherent problems: safe exploration and constraint bound tuning. To this end, through the use of metagradient optimization, the algorithm is capable of adjusting the safety-related hyperparameters with minimal initial tuning. Furthermore, the performance of the algorithm was studied in five

simulated embodied applications with the themes of locomotion, obstacle avoidance, robotic manipulation, and dexterous manipulation. It was observed that the synergy created between the parameters and the hyperparameters results in comparable or better performance of the policy in terms of reward or safety. Additionally, an experiment was conducted in a real-world setup involving a practical coffee-pouring robotic environment without any explicit safety-related reward shaping. The algorithm was deployed on the Kinova Gen3 robot and it was shown that the proposed algorithm can be helpful for real-world safety-sensitive applications by reducing the reliance on heuristic implementation of safety. It was also observed that formulation of safety solely as the violation rather than engineering the reward function results in applying lower levels of effort at the cost of a diminished success performance. This trade-off can be especially favorable in real-world setups where safety violations are costly. Specifically, the proposed algorithm will learn the optimal policy in the real-world setup while adhering to the collision constraints and minimizing the effort imposed on the robot.

Chapter 4

Safety Augmentation based on Multi-Objective Policy Optimization

4.1 Introduction

As discussed in the previous chapters, safe RL involves the process of maximizing the expected return of the policy while minimizing the expected probability of failure. To this end, the most common approach is to bound the constraint objectives and using constrained optimization methods to find the optimal solution. However, a major disadvantage of algorithms under this framework is their susceptibility to converge to a suboptimal policy due to suboptimal tuning of the safety-related hyperparameters.

To mitigate this, this chapter presents a novel model-free Safe RL algorithm, named Safety Optimized Reinforcement Learning (SORL), designed to enhance both safety and reward performance of the agent, simultaneously. Unlike the conventional methods, the safe RL problem is tackled as a multi-objective policy optimization

problem, which allows for the introduction of a reward-shaping technique that encourages the agent to explore the environment safely while striving to achieve better performance. This formulation provides an advantage over other model-free Safe RL approaches by eliminating the need to fine-tune the degree of constraining the policy search space (i.e., ε_{safe}). As a result, the algorithm will be able to reach a natural trade-off between performance and safety. Through the analysis of SORL, the safety of its converged policy is guaranteed through a condition. This condition motivates the introduction of the concept of aggressiveness in the algorithm which provides an intuitive way to tune the hyperparameters of the proposed algorithm. Furthermore, a constrained meta-optimization formulation based on the safety condition called Adaptive Safety Optimized Reinforcement Learning (ASORL) is proposed which allows the safety-related hyperparameter to be automatically optimized based onthe agressiveness parameter during the training process. Essentially, the proposed algorithm will be able to achieve safe performance with minimal safety-related hyperparameter tuning.

Finally, to demonstrate the effectiveness of the proposed algorithm, experiments are conducted in seven different safety-concerned simulated robotics problems, which are divided into three main safety topics, namely system-level safety, collision avoidance, and safe manipulation. The tasks are visualized in Figure 4.1. The algorithm is compared with six other state-of-the-art model-free Safe RL methods. The results indicate superior performance in both optimality and safety aspects.

As a summary, the technical contributions of this can be summarized as:

- A novel safety optimized reinforcement learning algorithm with minimal hyperparameter tuning requirements is proposed which optimizes the reward and safety performances simultaneously.

- A safety guarantee condition for the algorithm is proposed which offers an in-

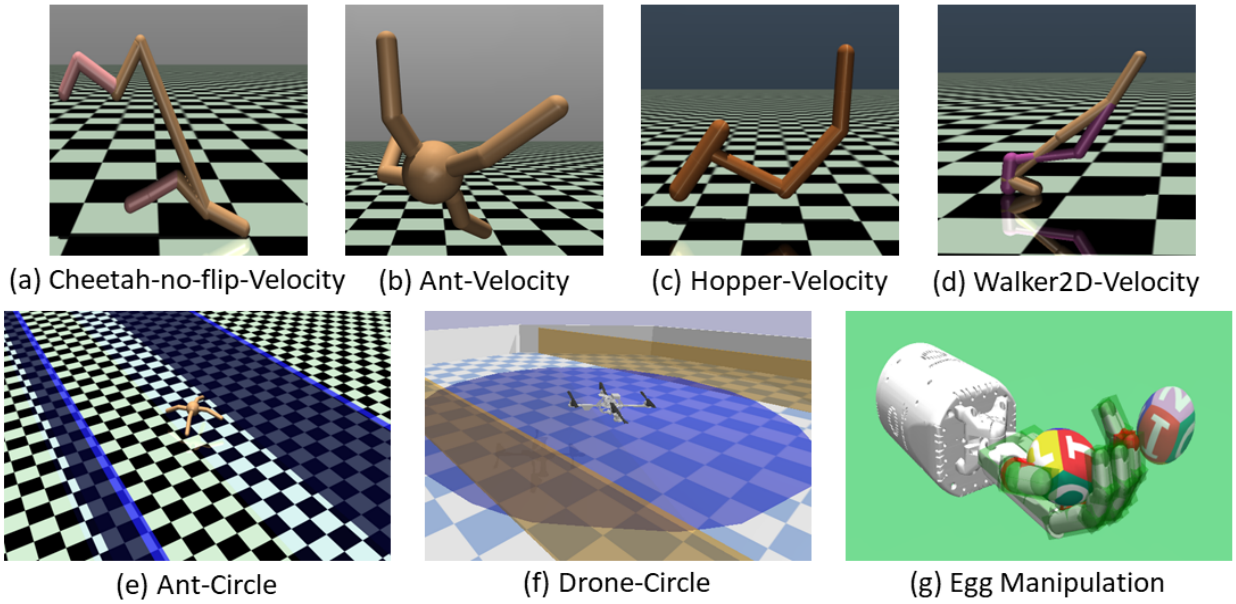


Figure 4.1: Visualization of the simulated safety-concerned robotics environments used to evaluate SORL in the context of three main safety topics of system-level safety (a-d), collision avoidance (e,f) and safe manipulation (g). The top row also showcases some possible constraint violation for the system-level safety benchmarks.

tuitive method to tune the safety-related hyperparameter of SORL.

- A constrained meta-optimization method is proposed to automatically tune the safety-related hyperparameter.
- The method's superior safety and reward performance is demonstrated across seven different robotics environments.

4.2 Methodology

In this section, a model-free safe reinforcement learning algorithm is proposed which uses safety signals to avoid unsafe regions. In the training process, the safety critic is used to modify the reward function such that the exploration in unsafe regions is prevented.

4.2.1 Reward Penalty Framework

Generally, the main purpose of safe RL algorithms is to identify, and avoid, the set of states that violate the safety constraints, which are referred to as S_{unsafe}. To do so, similar to the previous works on safe RL [54, 30], a subset of the state space are characterized that are not considered unsafe but will lead to an unsafe state inevitably:

Definition 1. *A state $s \in S$ is considered **Irrecoverable** if for any sequence of actions $a_0, a_1, a_2, ...$, starting from state $s_0 = s$ and following the transition probability $s_{t+1} \sim \mathcal{P}(s_t, a_t)\ \forall t \in \mathbb{N}$, there exists some time step $\bar{t} \in \mathbb{N}$ s.t. $s_{\bar{t}} \in S_{unsafe}$.*

Naturally, based on the definition, the safe state space S_{safe} encompasses the subset of the state space which are neither categorized as unsafe nor irrecoverable. Correspondingly, an action is considered safe if executing it leads to a safe state. Furthermore, the assumption can be made of a soon occurrence of safety violation after entering into it an irrecoverable state:

Assumption 1. *For any state $s_0 \in S_{\text{irrecoverable}}$ and for any sequence of actions $a_0, a_1, \ldots$ starting from s_0 and $s_{t+1} \sim \mathcal{P}(s_t, a_t)\ \forall t \in \mathbb{N}$, there exists $\bar{t} \in \{1, ..., H^*\}$* s.t. *$s_{\bar{t}} \in S_{\text{unsafe}}$.*

Finally, based on the reward penalty MDP $\langle S, A, P, \tilde{r}, \gamma \rangle$ which is introduced in [30], the reward function can be modified as:

$$\tilde{r}(s,a) = \begin{cases} r(s,a) & \text{if } s \notin S_{unsafe} \\ -C & \text{otherwise} \end{cases} \tag{4.1}$$

where C satisfies the following inequality:

$$C > \frac{r_{max} - r_{min}}{\gamma^{H^*}} - r_{max} \tag{4.2}$$

The terminal state cost $C \in \mathbb{R}$ is used to penalize the RL agent when unsafe trajectories are executed.

4.2.2 Multi-Objective Policy Optimization

Unlike the conventional Safe RL setting where the algorithm is trained under the Constrained MDP framework, SORL is formulated in a multi-objective policy optimization setting [55, 56]. To this end, *Safety-aware reward penalty MDP* is proposed $\langle S, A, P, \tilde{r}, c, \gamma, \gamma_{safe} \rangle$. Under this setting, in multi-objective policy optimization, there are multiple (often conflicting) objectives and the aim is to optimize the objectives simultaneously. For this purpose, the policy performance is defined as a 2-dimensional vector:

$$\mathcal{J}(\pi) = \begin{bmatrix} \mathcal{J}_r(\pi) \\ \mathcal{J}_c(\pi) \end{bmatrix} = \begin{bmatrix} \mathbb{E}^{\pi}_{\mathcal{P}}[\sum_{t=0}^{\infty} \gamma^t r_t] \\ \mathbb{E}^{\pi}_{\mathcal{P}}[\sum_{t=0}^{\infty} \gamma_c^t c_t] \end{bmatrix} \tag{4.3}$$

Furthermore, according to the Multi-Objective optimization literature, the dominance of a policy π relative to π' can be defined as:

$$\text{if} \quad \begin{cases} \mathcal{J}_r(\pi) \geq \mathcal{J}_r(\pi') \\ \quad \textbf{and} \\ \mathcal{J}_c(\pi) \leq \mathcal{J}_c(\pi') \end{cases} \Rightarrow \mathcal{J}(\pi) \succcurlyeq \mathcal{J}(\pi') \tag{4.4}$$

The field of multi-objective deep reinforcement learning is an active area of research that encompasses various methodologies aimed at optimizing the expected return of a set of potentially conflicting reward functions [57]. One widely-used approach for addressing this challenge involves scalarizing the reward vector using a scalarization function, which enables the optimization of the scalarized function and, subsequently, the optimization of all rewards. In the context of SORL, the objective

is to devise a reward-shaping scheme ($\hat{r}$) that optimizes both performance functions by optimizing a single policy:

$$\mathcal{J}_{\hat{r}}(\pi) \geq \mathcal{J}_{\hat{r}}(\pi') \Rightarrow \pi \succcurlyeq \pi' \tag{4.5}$$

4.2.3 Safety Optimized Reward Shaping

To address Equation 4.5, the notion of the reward function as a vector consisting of the actual reward function r and the safety signal function c seems plausible. However, it is not possible to use the safety signal directly in the scalarization function as it is a sparse function and its immediate value does not bear much significance. To this end, the augmented reward function is proposed, based on the safety estimate of the safety critic, defined as:

$$\tilde{r}(s_t, a_t) = \begin{cases} [1 - \lambda Q_c^{\pi}(s_t, a_t)] r(s_t, a_t), & \text{if } r(s_t, a_t) \geq 0 \\ \lambda Q_c^{\pi}(s_t, a_t) r(s_t, a_t), & \text{otherwise} \end{cases} \tag{4.6}$$

where $\lambda > 0$ is defined as the safety critic significance factor. Finally, under the safety-aware reward penalty MDP framework, the final reward shaping will be as follows:

$$\hat{r}(s_t, a_t) = \begin{cases} \tilde{r}(s_t, a_t), & \text{if } s_{t+1} \in S_{safe} \\ -C, & \text{otherwise} \end{cases} \tag{4.7}$$

4.2.4 Safety Guarantee

The aim of this section is to guarantee the safety of the converged policy when Equation 4.7 reward shaping scheme is used which will help during the hyperparameter tuning phase. To prove the theorem, the following assumptions are used:

Assumption 2. *The reward function r is bounded in the range $[r_{min}, r_{max}]$ where $r_{min} < 0$ and $r_{max} > 0$.*

Assumption 3. *The environment is deterministic in terms of the safety violations. In other words, for any state $s \in S$, $\Pr[c(s) = 1]$ is either 0 or 1.*

In order to provide a safety guarantee the irrecoverable states discussed in Section 4.2.1 must first be studied. Using Assumption 1 allows for further categorizing the irrecoverable states into levels of unsafety.

Lemma 1. *In an environment where Assumption 1 holds, for any trajectory $\tau = \{(s_0, a_0), ..., (s_{|\tau|}, a_{|\tau|})\}$ where $s_0 \in S_{irrecoverable}$, $s_t \sim \mathcal{P}(s_{t-1}, a_{t-1})$, and $s_{|\tau+1|} \sim \mathcal{P}(s_{|\tau|}, a_{|\tau|}) \in S_{unsafe}$, and for any $t \in \{1, ..., |\tau|\}$ we have:*

$$Q_c^{\pi}(s_t, a_t) \geq (\gamma_c)^{H^* - t} \tag{4.8}$$

Proof. Recall that from Assumption 1 the length of the trajectory is $|\tau| \leq H^*$. Moreover, at any time step t and its corresponding state s_t, consider the space of all the trajectories that start from s_t: $\mathrm{T}_t = \{\tau' : (s'_0 = s_t, a'_0), \ldots, (s'_{|\tau'|}, a'_{|\tau'|}), s'_{|\tau'+1|} \sim \mathcal{P}(s_{|\tau'|}, a_{|\tau'|}) \in S_{unsafe}\}$. Suppose there exists a trajectory $\tau' \in \mathrm{T}_t$ with $|\tau'| > H^* - t$. Consequently, the concatenation $\tau(s_0 : s_t) \cup \tau'$ will have a length greater than H^* which will violate Assumption 1 since there will then exist a trajectory from the irrecoverable state s_0 with a length greater than H^*.

Therefore, based on the original trajectory τ, it can be concluded that $\max_{\tau' \in \mathrm{T}_t} |\tau'| \leq H^* - t$. Finally, by the definition of the safety critic in Equation 2.34, due to Assumption 3 which will dictate the point at which the trajectory and the summation ends, it can be concluded that the summation continues for at most $H^* - t$ steps. Therefore, $(\gamma_c)^{H^* - t}$ will be the lowest value of the safety critic. □

It is noteworthy that to establish safety bounds for the safety critic's evaluation in safe states, it is needed to make assumptions about the policy's behavior and its interactions with the environment because in some cases, the policy may choose unsafe actions even in safe states, making the safety critic's output an expectation that includes both safe and unsafe actions. Therefore, to maintain a broad analysis, such limiting assumptions are avoided.

Theorem 1. *Under the safety-aware reward penalty MDP framework, let the following condition for safety critic significance factor λ hold:*

$$\frac{r_{max}}{1-\gamma} - \frac{r_{max}+C}{1-\gamma}\gamma^{|\tau^*_{uwc}|} < \left(\frac{\gamma_c^{H^*} r_{max}}{1-\frac{\gamma}{\gamma_c}} - \frac{\gamma_c^{H^*} r_{max}}{1-\frac{\gamma}{\gamma_c}}\left(\frac{\gamma}{\gamma_c}\right)^{|\tau^*_{uwc}|} + \frac{\gamma_c r_{min}}{1-\gamma}\right)\lambda \tag{4.9}$$

where C and $|\tau^*_{uwc}|$ follow Equation 4.2 and Equation 4.12, respectively. Consequently, for any state s: $\hat{Q}^*(s,a) > \hat{Q}^*(s,a')$, where action a is safe, a' is unsafe, and $\hat{Q}^*$ is the Q^* value-function following Equation 4.7 reward-shaping.

Proof. By Assumption 1, the trajectory starting from an unsafe action a' will have a length of at most H^* before constraint violation. Therefore, by Assumption 2, assuming that the agent will remain in the unsafe state for the rest of the episode, the maximum discounted return in the worst-case scenario can be expressed as a function of the length of the trajectory before reaching the unsafe state:

$$R^{\pi}_{wc}(|\tau|) = \sum_{t=0}^{|\tau|-1}\left(\gamma^t\left[1-\lambda Q^{\pi}_c(s_t,a_t)\right]r_{max}\right) + \sum_{t=|\tau|}^{\infty}\left(\gamma^t(-C)\right) \tag{4.10}$$

While it is possible to ignore the safety critic term and upper bound the equation, a tighter upper bound is presented in this case. First, Lemma 1 is used to upper bound

the function (in the following we use the notation x to indicate the variability of $|\tau|$):

$$\begin{aligned} R^{\pi}_{wc}(x) &\leq \sum_{t=0}^{x-1}(\gamma^t\left[1-\lambda\gamma_c^{H^*-t}\right]r_{max}) + \sum_{t=x}^{\infty}(\gamma^t(-C)) \\ &= (\frac{r_{max}}{1-\gamma} - \frac{\lambda\gamma_c^{H^*}r_{max}}{1-\frac{\gamma}{\gamma_c}}) - \frac{r_{max}+C}{1-\gamma}\gamma^x \\ &\qquad + \frac{\lambda\gamma_c^{H^*}r_{max}}{1-\frac{\gamma}{\gamma_c}}(\frac{\gamma}{\gamma_c})^x = R_{uwc}(x) \end{aligned} \tag{4.11}$$

To find the maximum unsafe return in the domain $x \in [1, H^*]$, the derivative with respect to x is taken and set to zero: $\partial R^{\pi}_{uwc}(x)/\partial x = 0$. By solving this derivative and ensuring that the trajectory length remains within the range $|\tau_{uwc}| \in [1, H^*]$, the trajectory length with the highest return can be determined as:

$$|\tau^*_{uwc}| = \begin{cases} \left\lfloor \dfrac{\ln(\dfrac{\lambda\gamma_c^{H^*}r_{max}}{1-\dfrac{\gamma}{\gamma_c}}\ln\dfrac{\gamma}{\gamma_c}) - \ln(\dfrac{r_{max}+C}{1-\gamma}\ln\gamma)}{\ln\gamma_c} \right\rfloor & \text{if} \in [1, H^*] \\ \underset{|\tau|\in\{1,H^*\}}{\operatorname{argmax}} R_{uwc}(|\tau|) & \text{otherwise} \end{cases} \tag{4.12}$$

Hence, Equation 4.10 can be upper bounded as:

$$R^{\pi}_{wc}(|\tau|) \leq R_{uwc}(|\tau^*_{uwc}|) \tag{4.13}$$

Furthermore, executing the safe action a leads to a safe state where a safe trajectory can be generated which does not encounter a safety violation. The discounted return

with the minimum reward (Assumption 2) can be lower bounded as:

$$\begin{aligned}\sum_{t=0}^{\infty}(\gamma^t \lambda Q_c^{\pi}(s_t, a_t) r_{min}) &= \lambda r_{min} \sum_{t=0}^{\infty}(\gamma^t Q_c^{\pi}(s_t, a_t)) \\ &\geq \lambda r_{min} \gamma_c \sum_{t=0}^{\infty}(\gamma^t) = \frac{\lambda \gamma_c r_{min}}{1-\gamma}\end{aligned} \tag{4.14}$$

To establish the inequality, the observation that $Q_c^{\pi}(s_t, a_t) \leq \gamma_{safe}$ is leveraged. This is based on the fact that for any timestep t, execution of the state-action pair (s_{t-1}, a_{t-1}) leads to the safe state s_t. Hence, Assumption 3 ensures that $\Pr[c(s_t) = 1] = 0$; consequently, the right hand side equation in Equation 2.35 ($Q_c^{\pi}(s, a) = \Pr[c(s) = 1] + \gamma_c \mathbb{E}_{\mathcal{P}}^{\pi}\left[(1 - c(s)) Q_c^{\pi}(s', a')\right]$) can be simplified as $\gamma_{safe} \mathbb{E}_{\mathcal{P}}[V_c^{\pi}(s')]$. Finally, since the expectation operator outputs a value within the range of zero and one, the observation is justified.

Therefore, since the discounted return of staying safe forever must always be higher than a trajectory that has a safety violation, it suffices that:

$$\Delta = \frac{\lambda \gamma_c r_{min}}{1-\gamma} - R_{uwc}(|\tau_{uwc}^*|) > 0 \tag{4.15}$$

Rearranging Equation 4.15 gives the condition. □

It is possible to derive bounds similar to Theorem 1 with other variations of Assumption 2. Moreover, intuitively, the value of Δ in Equation 4.15 determines the degree of aggressiveness of the algorithm. The main difference between Δ and the threshold in CMDP-based methods is the fact that in CMDP methods, the threshold first specifies the degree with which the policy is constrained; following that, the constrained optimization formulation determines the tradeoff between the reward objective and the constraint objective. However, using the formulation of Δ allows for direct specification of the tradeoff between the two objectives. The closer this

Algorithm 2 Safety Optimized RL

Require: Safety critic significance factor λ and Δ, horizon H^*

Initialize policy π_θ, safety-augmented critic $\hat{Q}_{\omega_1}, \hat{Q}_{\omega_2}$, replay buffer $\mathcal{D}$, safety critic $Q_c^{\omega_1}, Q_c^{\omega_2}$, replay buffer $\mathcal{D}_{safe}$
for $e = 1, ..., E_{max}$ **do**
    $s_1 \leftarrow env.reset()$
    for $t = 1, ..., T_{max}$ **do**
        $a_t \sim \pi_\theta(.|s_t)$
        $\hat{c}_t \leftarrow \max\{Q_c^{\omega_1}(a_t|s_t), Q_c^{\omega_2}(a_t|s_t)\}$
        $s_{t+1}, r_t, c_t, done \leftarrow env.step(a_t)$
        compute empirical r_{min}, r_{max} and update C
        solve and update λ (Eq. 4.9)
        if $c_t == 0$ **then**
            if $r_t > 0$ **then** $\hat{r}_t \leftarrow \left[1 - \lambda \hat{c}_t\right] r_t$
            else $\hat{r}_t \leftarrow \lambda \hat{c}_t r_t$
        else
            $\hat{r}_t \leftarrow -C$
            $\mathcal{D}_{safe} \leftarrow \mathcal{D}_{safe} \cup (s_t, a_t, c_t, \hat{r}_t, s_{t+1})$
        $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, c_t, \hat{r}_t, s_{t+1})$
        train $\pi_\theta, \hat{Q}_{\omega_1}, \hat{Q}_{\omega_2}$ on $\mathcal{D}$
        train $Q_c^{\omega_1}, Q_c^{\omega_2}$ on $\mathcal{D} \cup \mathcal{D}_{safe}$
        if done **then** Break

value is to zero, the more the algorithm prioritizes optimality and performance over its failure rate. Hence, for each task, the level of aggressiveness in SORL is defined by specifying the value of Δ and fine-tuning λ to align it as closely as possible with the specified value.

4.2.5 Meta-Gradient Optimization

Fundamentally, λ is a hyperparameter that specifies the trade-off between safety and optimality and directs the optimization of the policy based on that. Hence, in the Pareto front of optimality vs. safety, the choice of λ dictates where the performance will lie on the Pareto front. However, optimizing the value of λ cannot be done using the traditional algorithms such as grid search [58] or genetic algorithm [59] since, for

the case of a reinforcement learning application, using them is computationally expensive because it would need to the policy to be rolled-out from the beginning based on each value of λ and compare the results with each other; therefore, rendering it impossible to use in real-time. To address this issue, a constrained meta-optimization approach is proposed to automatically optimize the value of λ. As discussed in Section 2.4, the basic idea of these methods is the bilevel optimization framework. As explained in [32] for the context of RL, in this framework the internal parameters, i.e. the policy, are optimized based on the internal objective function. In other words, the updated policy parameter θ can be written as:

$$\theta' \leftarrow \theta + f(\theta, \lambda, \mathcal{B}) \tag{4.16}$$

where $\mathcal{B}$ is a batch of experiences to evaluate gradient w.r.t. the safety-augmented critic network parameterized as ω:

$$f(\theta, \lambda, \mathcal{B}) = \beta_\theta \nabla_\theta \, \mathbb{E}_{\substack{\{s\}=\mathcal{B}\sim\mathcal{D} \\ a\sim\pi_\theta}}[\hat{Q}_\omega^{\pi_\theta}(s, a)] \tag{4.17}$$

Here, the inner objective is specified to be $\hat{Q}_\omega$ the value function following Equation 4.7.

Furthermore, the effect of λ can be evaluated based on the meta-objective $\mathcal{J}_\lambda$:

$$\frac{\partial \mathcal{J}_\lambda(\theta', \lambda, \mathcal{B}')}{\partial \lambda} = \frac{\partial \mathcal{J}_\lambda(\theta', \lambda, \mathcal{B}')}{\partial \theta'} \frac{\mathrm{d}\theta'}{\mathrm{d}\lambda} \tag{4.18}$$

In order to evaluate the derivative of θ' w.r.t. λ, it needs to have differentiable effect on the policy update. However, since the augmentation happens on the level of the reward function, determining the effect on the level of the value function can be challenging. Hence, to determine the effect of λ, as proposed in [32], in this research,

the inner policy objective function is conditioned on the value of λ:

$$\mathcal{J}_{\pi}(\pi_{\theta}, \lambda) = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{\theta}}} \left[\hat{Q}_{\omega}^{\pi_{\theta}}(s, \pi_{\theta}(s); \mathbf{e}_{\lambda})\right] \tag{4.19}$$

Where $\mathbf{e}_{\lambda}$ is defined as the embedding for λ:

$$\mathbf{e}_{\lambda} \leftarrow \mathbf{W}_{\lambda}\lambda \tag{4.20}$$

where $\mathbf{W}_{\lambda}$ is a learnable embedding vector. The use of embedding as input to the value network provides more stability in the training process of the algorithm. To address the conditioning, the Bellman equation of the safety-augmented critic for non-terminal states is specified as:

$$\hat{Q}_{\omega}^{\pi_{\theta}}(s, a; \mathbf{e}_{\lambda}) = \max\left(r(s, a), 0\right) - \lambda Q_{c}^{\pi_{\theta}}|r(s, a)| + \gamma\hat{Q}_{\omega}^{\pi_{\theta}}(s', a'; \mathbf{e}_{\lambda}) \tag{4.21}$$

4.2.6 Adaptive SORL

In this section, the process of optimizing the λ value is discussed. To this end, the optimization scheme of λ is defined as a constrained optimization formulation:

$$\lambda^{*} = \underset{\lambda>0}{\operatorname{argmax}}\ \mathcal{J}_{\lambda}(\pi_{\theta'}) \text{ s.t. } \Delta(\lambda) > 0 \tag{4.22}$$

By having $\Delta(\lambda) > 0$ during the training process the convergence of the policy enjoys the guarantee of safety. Furthermore, while the Lagrangian relaxation technique [60, 18] for Equation 4.22 can be applied to optimize λ, the introduction of additional hyperparameters such as the Lagrangian multiplier can pose challenges of tuning. This challenge was observed in the early experiments with the Lagrangian formulation. The optimization process was heavily influenced by this value and could

easily lead to suboptimal behaviors. To avoid this issue, since the initial value of λ can easily be chosen such that it satisfies the safety guarantee, the constraint $\Delta(\lambda) > 0$ is treated as a hard constraint. Hence, in each step of meta-optimization of λ, the new value of λ is accepted if the safety guarantee condition is satisfied:

$$\lambda_{k+1} \leftarrow \lambda_k + \beta_\lambda \mathbb{1}[\Delta(\lambda_{k+1}) > 0] \nabla_\lambda \mathcal{J}_\lambda(\pi_{\theta'}) \tag{4.23}$$

The gradient rejection process is valid because in most cases $\Delta(\lambda)$ is a convex constraint. For that reason, starting the process from an initial value that satisfies the constraint must result in obtaining the optimal value without violating it. The convexity stems from the fact that in almost all of the cases, $|\tau^*_{uwc}| = H^*$ because in most cases the longest unsafe worst-case trajectory would result in the highest worst-case return. Hence, by having $|\tau^*_{uwc}| = H^*$ (or any other constant), $\Delta(\lambda)$ would become linear and convex.

Finally, in order to optimize the value of λ, $J_\lambda(\pi_{\theta'})$ is needed to be defined. Fundamentally, since the safety of the policy is guaranteed through the condition of Δ, λ simply needs to be adjusted such that it maximizes the expected return of the policy. Hence, it suffices that the meta-objective be defined as the value-function of the original reward function of the environment:

$$\mathcal{J}_\lambda(\pi_{\theta'}) = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{\theta'}}} \left[Q^{\pi_\theta}_{\omega_r}(s, a) \right] \tag{4.24}$$

The choice of this formulation stems from the experiments done with different variations of this objective function and have been discussed as ablation in Section 4.3. The other variations experimented with are based on the policy being deterministic or stochastic and the states coming from $\mathcal{D}$ or $\mathcal{D}_0$ (initial state buffer).

The pseudocode of the Adaptive SORL training process is discussed in Algo-

Algorithm 3 Adaptive Safety Optimized RL

Require: Safety critic significance factor λ and Δ, horizon H^*

Initialize policy π_θ, critic $Q_r^{\omega_1}, Q_r^{\omega_2}$, safety-augmented critic $\hat{Q}_{\omega_1}, \hat{Q}_{\omega_2}$, replay buffer $\mathcal{D}$, safety critic $Q_c^{\omega_1}, Q_c^{\omega_2}$, replay buffer $\mathcal{D}_{safe}$
for $e = 1, ..., E_{max}$ **do**
  $s_1 \leftarrow env.reset()$
  for $t = 1, ..., T_{max}$ **do**
    $a_t \sim \pi_\theta(.|s_t)$
    $\hat{c}_t \leftarrow \max\{Q_c^{\omega_1}(a_t|s_t), Q_c^{\omega_2}(a_t|s_t)\}$
    $s_{t+1}, r_t, c_t, done \leftarrow env.step(a_t)$
    compute empirical r_{min}, r_{max} and update C
    if $c_t == 1$ **then**
      $r_t \leftarrow -C$
      $\mathcal{D}_{safe} \leftarrow \mathcal{D}_{safe} \cup (s_t, a_t, c_t, r_t, s_{t+1})$
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, c_t, r_t, s_{t+1})$
    train $Q_c^{\omega_1}, Q_c^{\omega_2}$ on $\mathcal{D} \cup \mathcal{D}_{safe}$
    Sample a batch of transitions $\mathcal{B} = \{(s, a, r, c, s')\} \in \mathcal{D}$
    train $Q_r^{\omega_1}, Q_r^{\omega_2}, \hat{Q}_{\omega_1}, \hat{Q}_{\omega_2}$ using $\mathcal{B}$ (Eq. 4.21)
    $\theta' \leftarrow \theta + \beta_\theta \nabla_\theta \mathcal{J}_\theta(\pi_\theta, \lambda)$ using $\mathcal{B}$ (Eq. 4.19)
    Resample $\mathcal{B}' = \{s \in \mathcal{D}\}$
    $\lambda' \leftarrow \lambda + \beta_\lambda \nabla_\lambda \mathcal{J}_\lambda(\pi_{\theta'})$ using $\mathcal{B}'$ (Eq.4.24)
    if $\Delta(\lambda') > 0$ **then**
      $\lambda \leftarrow \lambda'$
    $\theta \leftarrow \theta'$
    if done **then** Break

rithm 3.

4.3 Experimental Results

In the following section, the performance of (A)SORL is studied. Particularly, the aim is to investigate two questions:

- How does the safety formulation performs compared with the other model-free off-policy Safe RL algorithms?

- How does the value of Δ in Equation 4.15 affect the performance of (A)SORL?

4.3.1 Benchmarks and Comparison Methods

In order to evaluate the level of safety that the model can achieve, it is executed on three safety-concerned categories of environments:

- **System-level safety:** RL algorithms are often used to optimally control robots while adhering to the system limits. The proposed model is assessed using four MuJoCo environments: *Cheetah-no-flip-Velocity, Ant-Velocity, Hopper-Velocity,* and *Walker2D-Velocity.* In these environments, the agent must learn to move faster in the x-direction while avoiding actions that cause the robot to fall and fail. Additionally, safety violations occur if the robot exceeds a certain velocity. The codebase is obtained for Hopper-Velocity, Walker2D-Velocity, and Ant-Velocity from [40]. For Cheetah-no-flip-Velocity, the base environment was adopted from [30] and the Velocity constraint was added to it.

- **Collision Avoidance:** Besides the inherent robot limits, additional constraints from the environment can impact the algorithm. Collision avoidance is one such constraint, where the controller aims to control the robot while preventing collisions with obstacles. The algorithms are evaluated on this aspect using Ant-Circle [16] and Drone-Circle (from Bullet Safety Gym codebase [61]) environments. These assessments involve controlling robots to move in circular paths while staying within a safety region smaller than the circle's radius.

- **Safe Manipulation:** Finally, one of the important applications of Safe RL is safe manipulation. To this end, a modified version of the in-hand object manipulation is adopted from Gymnasium Robotics [43] which uses a dexterous hand to manipulate an egg to achieve a target pose. In this task, if the hand

exerts a normal force more than a threshold (20 N), the egg will get crushed and the agent will fail.

The episode ends whenever a safety violation has been incurred. Moreover, in all the environments, the alive bonus has been eliminated to evaluate the performance of the safety algorithms in situations where the original reward shaping does not explicitly encode safety. An overall description of the task environments are discussed in Table 4.1.

Table 4.1: Information about the environments used.

Task name	**dim($\mathcal{S}$)**	**dim($\mathcal{A}$)**	**Description**
Cheetah-no-flip [62]	17	6	Control a 2D HalfCheetah locomotion robot to move forward without exceeding a certain velocity and falling over to the ground.
Ant-Velocity [63]	27	8	Control the legs of a 3D ant robot to move in the forward direction without exceeding a certain velocity and keeping the head within a specific height.
Hopper-Velocity [64]	11	3	Control a 2D one-legged robot to hop in the forward direction while keeping it healthy without exceeding a certain velocity.
Walker2D-Velocity [64]	17	6	Control a 2D two-legged robot to move in the forward direction while keeping it healthy without exceeding a certain velocity.
Ant-Circle [63, 16]	27	8	Control the Ant robot to move in a circle while avoiding collision with two walls and keeping the head within a specific height.
Drone-Circle [45]	17	4	Control a Drone to hover in a circle path while avoiding collision with two walls.
Egg Manipulate [47, 46]	153	20	Control a Shadow Dexterous Hand to manipulate an Egg to reach a goal orientation without exerting force more than 20N.

Six model-free Safe RL algorithms are used to showcase the performance of SORL and ASORL. The comparison algorithms include Lagrangian Relaxation (LR), Safety

Table 4.2: Hyperparameter values and architecture details of the methods

Parameter	Value
γ	0.99
γ_c	0.5 for Cheetah-No-Flip 0.6 for Other Environments
H^*	10
Embedding Size ($\mathbf{W}_\lambda$)	16
λ Initial Value	1
λ Gradient Norm Threshold	5e−4
Polyak Averaging Factor (τ)	5e−3
Hidden Layer Size	$[256, 256]$
Number of Hidden Layers	3
Policy & Critic Network Learning Rate	1e−4
Activation Function	GELU for ASORL ReLU for Other Methods
λ learning rate (β_λ)	5e−5
Replay Buffer Size	1e6
Batch Size	256

Q-Functions for RL (SQRL) [25], Model-Free Recovery RL (RRL-MF) [31], Risk Sensitive Policy Optimization (RSPO) [65], and Reward Constrained Policy Optimization (RCPO) [18]. Finally, to study the safety performance of (A)SORL reward shaping, SAC+C is executed which uses Equation 4.1 reward scheme.

4.3.2 Implementation Settings

The codebase for the comparison methods are adopted from [31] codebase, and, to have a fair comparison between the algorithms and evaluate their safety, pretraining is disabled for all of the algorithms. To this end, (A)SORL and the comparison methods are built on top of the Soft Actor-Critic algorithm [13], and, for fair comparison, the general and common hyperparameters of all the algorithms are kept the same. In addition to that, with the help from the problem-specific hyperparamter settings discussed in [30], the parameters of the comparison algorithms are tuned for each benchmark problem. Moreover, for the hyperparameters relating to (A)SORL, $H^* =$

10 is set for all the cases and tune the proposed algorithm based on the value of Δ. For each task, the value of Δ used to execute SORL is shown in Figure 4.3. During the experimentation of various Δ values within the specified environments, significant variations in the performance of SORL was observed when its change of value is near the magnitude of 50. The magnitude of change in Δ can be largely attributed to the choice of γ and γ_{safe}. In addition, the value of C in Equation 4.7 follows Equation 4.2 and it is chosen as the nearest value to the lower bound by empirically calculating the values of r_{min} and r_{max}.

Furthermore, for training ASORL, the initial value $\lambda = 1$ is set for all of the tasks. Moreover, the changes in λ can have drastic effects on the stability of the systems. Hence, in order to promote stability in the optimization process the overall gradient magnitude of $\log \lambda$ (which is the optimization variable used in the implementation codebase) is restricted to 0.0005. Finally, it is important to note that while other methods mentioned and implemented in this thesis (even metagradient-based ones in Chapter 3) have utilized the ReLU (Equation 2.11) as their activation function, ASORL uses GELU as the activation function for all of the neural networks. This choice stems from the second-order derivative of these functions. Since the effect of λ is captured using the neural network, a second-order gradient of the network is needed to be calculated for performing meta-optimization. Hence, since the second-order derivative of ReLU is zero, the algorithm would output zero for the metagradient of λ. In contrast, in other methods, such as Meta SAC-Lag in Chapter 3, since the meta-optimization variables were directly present in the objectives of the other parameters, the optimization was possible even with a ReLU activation function.

The important architecture and hyperparameter details are included in Table 4.2. Furthermore, it is noteworthy that the results in the following sections illustrate the mean and variance of the execution of the algorithms with independent random seeds.

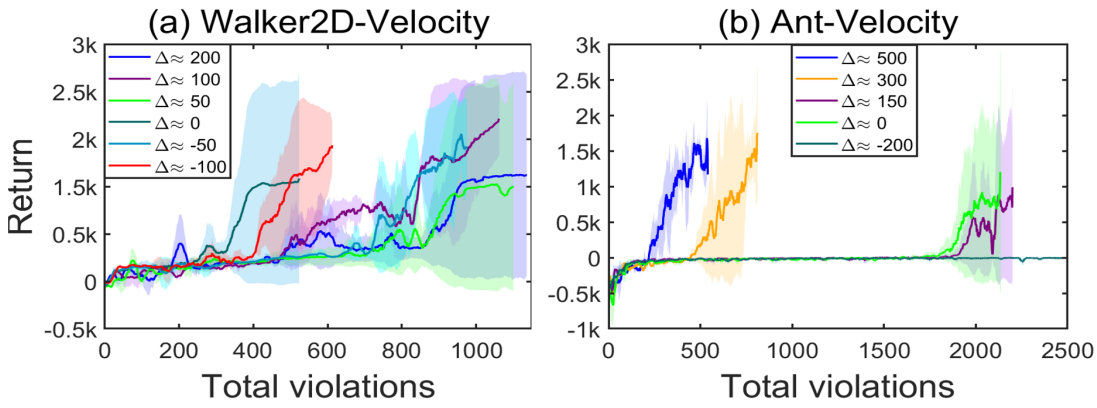


Figure 4.2: Undiscounted return of the policy versus the total number of violations during the training phase.

4.3.3 Effect of Δ on SORL

In this section, the effect of Δ on the performance of SORL is studied in two environments. The undiscounted return of the policy achieves whenever a safety violation has occurred is reported which can show sample efficiency of the algorithm in terms of safety. To gain a better understanding of the effect of Δ, negative values were also chosen to study its performance under too aggressive specifications. As depicted in Figure 4.2, while being aggressive in Walker2D-Velocity helps SORL attain higher returns in lower number of constraint violations, better performance in Ant-Velocity requires more conservativeness. This may be due to the difference in the dynamics of the robots and their constraints since Ant-Circle also requires a more conservative Δ value. The dynamics differences is more evident in the comparison between $(-100, -50)$ (where there is no safety guarantee) with $\Delta \approx (500, 300)$ in their respective tasks which indicates while being a little more aggressive in one helps in the reduction of the number of constraint violations, the opposite holds true for the other task. Finally, the superior performance of SORL in specific Δ values for their respective task motivates the use of ASORL to automatically tune λ based on this value.

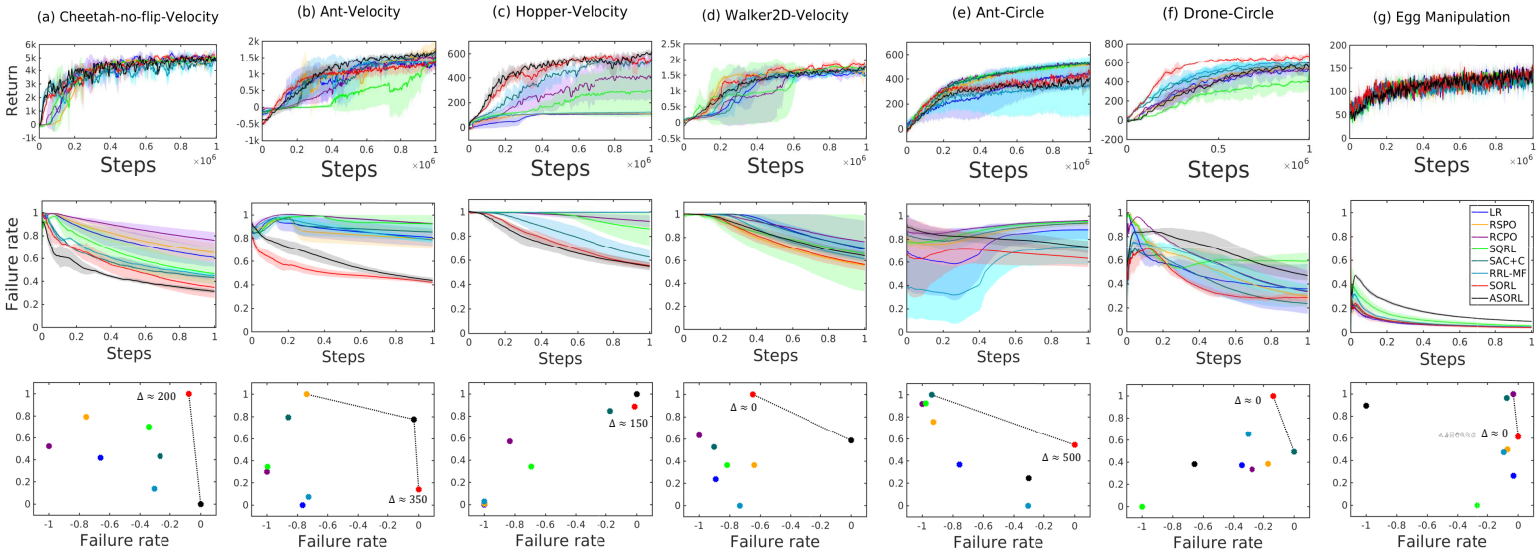


Figure 4.3: Benchmark results of (A)SORL compared with six other Safe RL algorithms. **(Top row):** Return values achieved during the training phase (higher is better). **(Middle row):** Episodic failure rates suffered during the training phase (lower is better). **(Bottom row):** Pareto optimality plot corresponding to the return and failure rate values (closer to the top right corner is better). They also display the specific value of Δ employed for executing SORL. For easier comparison, the return and failure rate values are normalized and the failure rate is scaled to lie within -1 and 0. The Pareto optimality solutions are highlighted through the dotted line.

4.3.4 Experiment Results

For the performance comparison of the different Safe RL algorithms, as depicted in Fig. 4.3, the reward performance and the failure rate of the algorithms are reported. Based on the learning curves, the pareto optimality of the algorithms based on Equation 4.5 are plotted. The results show dominant and superior performance of SORL or ASORL in both aspects of safety and optimality in Cheetah-no-flip-Velocity, Hopper-Velocity, and Walker2D-Velocity. Furthermore, in Ant-Velocity, Ant-Circle, and Egg Manipulation it can be observed that the proposed methods attain better safety per-

formance while achieving comparable returns. Importantly, the results in Ant-Circle indicate that while the comparison algorithms attain high returns, their policy is suboptimal in terms of the safety performance. Noticably, the suboptimality of the converged policies of the comparison methods in Hopper-Velocity and Ant-Circle in one or both aspects of performance can be seen. Finally, the results in Walker2D-Velocity and Drone-Circle illustrate SORL's consistently higher returns while also maintaining near-dominant safety performance. Regarding the comparison between the hyperparameter-tuned SORL and Adaptive SORL, it can be observed that the meta-gradient optimization scheme provides a powerful formulation for optimizing the trade-off between optimality and safety. In cheetah-no-flip-velocity, Ant-Velocity, Hopper-Velocity, and Walker2D-Velocity ASORL is among the well performing algorithm in the Pareto front, On the other hand, in Ant-Circle, Drone-Circle, and Egg Manipulation the performance of ASORL is dominated by other algorithms. This phenomenon can be attributed to the fact that in these environments the optimization of λ would require lots of samples; therefore, the performance would become diminished compared to the other algorithms. In summary, it can be concluded that (A)SORL can strike a great balance between safety and optimality offering a novel solution for safe performance among Safe RL algorithms.

4.3.5 Meta-Objective Ablation Study

In this section the performance of the meta-objective for λ in Equation 4.24. To this end, four main variations of the meta-objective are studied:

1. $\mathcal{J}_\lambda(\mathcal{D}_0, \pi_{det})$: Uses the initial state distribution and the deterministic mean of the action distribution to evaluate the objective function.

2. $\mathcal{J}_\lambda(\mathcal{D}_0, \pi_{stochastic})$: Uses the initial state distribution with the action sampled

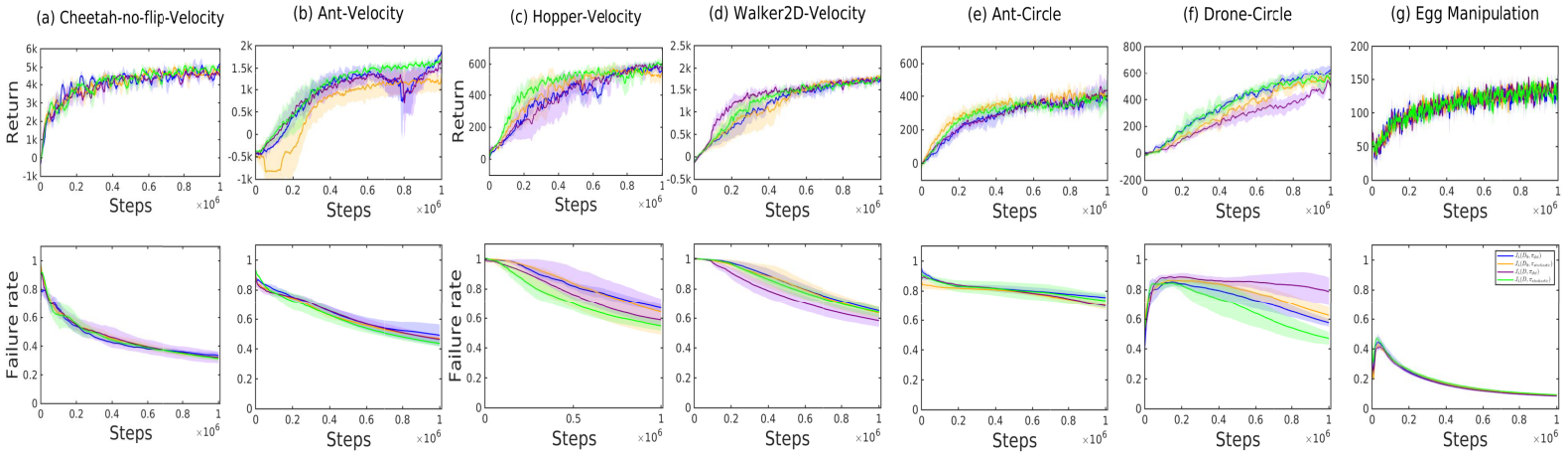


Figure 4.4: Comparison results of four variations of the meta-objective of ASORL.**(Top row):** Return values achieved during the training phase (higher is better). **(Middle row):** Episodic failure rates suffered during the training phase (lower is better).

from the policy action distribution.

3. $\mathcal{J}_\lambda(\mathcal{D}, \pi_{det})$: Uses all the visited states distribution along with the deterministic mean of the action distribution to evaluate the objective function.

4. $\mathcal{J}_\lambda(\mathcal{D}, \pi_{stochastic})$: The variation proposed in Equation 4.24. It uses all the visited states distribution and the action sampled from the policy action distribution.

The experimentation results are depicted in Figure 4.4. The results indicate that while in some tasks such as Walker2D-Velocity, Ant-Circle, Cheetah-no-flip-Velocity, and Egg Manipulation all the meta-objectives achieve similar results, in Drone-Circle and Hopper-Velocity the proposed meta-objective achieves better result. However, the results in Ant-Velocity indicate better training stability of the algorithm while utilizing all of the observed states in the buffer for evaluation of the meta-objective.

Furthermore, the changes in the value of Δ and λ are reported in Figure 4.5. It can be observed that in the cases of Ant-Velocity, Hopper-Velocity, Cheetah-no-flip-Velocity, and Ant-Circle the value of Δ is relatively close to the hyperparameter-tuned

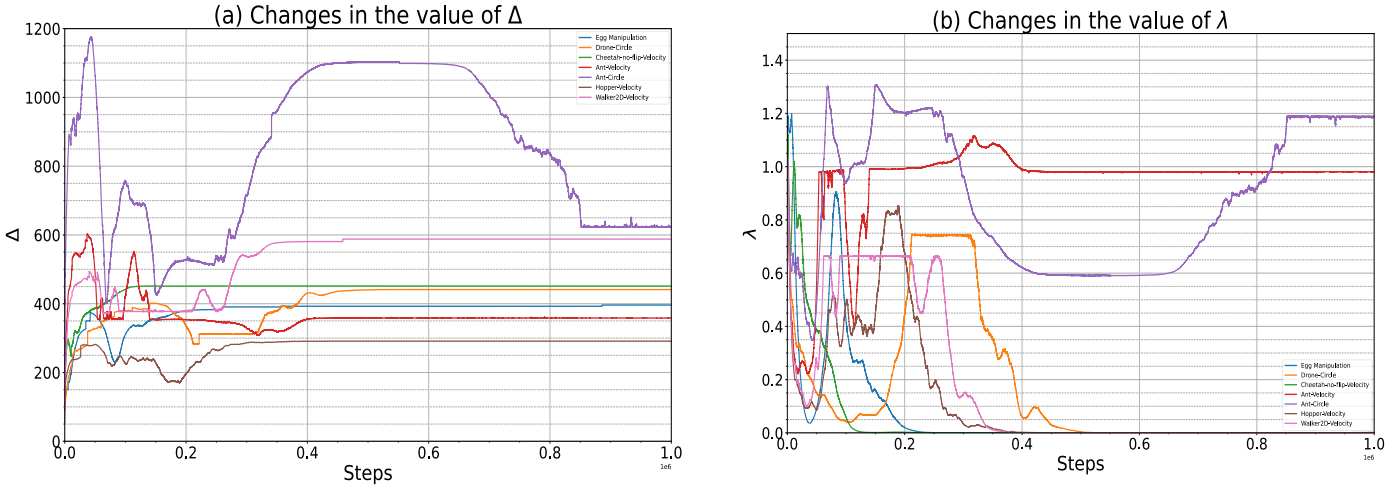


Figure 4.5: The changes in the value of Δ and λ during the training process of ASORL.

values in SORL. Moreover, while the changes in λ are volatile, it can be observed that the changes in the value are aimed at convergence to a certain level of aggressiveness of Δ. The volatility in the changes of the value of λ can be mainly attributed to the conditioning of the critic neural network on the value of λ (Equation 4.21). The optimization process can be susceptible to various phenomena such as catastrophic forgetting [66], inaccurate approximation, or other possible biases in the model.

4.4 Conclusions

This chapter focused on the problem of safe exploration and decision-making for RL agents. A novel Safe RL augmentation approach based on multi-objective policy optimization framework was proposed which optimized the policy toward optimality and safety, simultaneously. Through theoretical analysis, the safety of (A)SORL's converged policy was guaranteed through a condition which allowed the introduction of the concept of aggressiveness. The concept provided an intuitive way to tune SORL's safety-related hyperparameter. Furthermore, the use of the safety guarantee condition and the meta-gradient optimization allowed automated tuning of the afore-

mentioned trade-off to require minimal tuning of the safety-related hyperparameters. Finally, three main safety topics (viz., system-level safety, collision avoidance, and safe manipulation) were studied through seven different tasks in total. The reward and safety performance of the proposed algorithm was evaluated against six other state-of-the-art model-free Safe RL approaches. The results showed great capability of the safety augmentation in attaining better safety performance while achieving better or comparable returns. While the tuning of the safety-related hyperparameter demonstrated a consistent good performance of SORL on a wide range of robotic applications, the constrained meta-optimization method provided for ASORL resulted in better or competitive performance in certain tasks and suboptimal performance in the other ones. The main cause of the suboptimal performance can be mainly attributed to the reliance on neural network approximation in the meta-optimization process.

Chapter 5

Conclusions

5.1 Concluding Remarks

This research focused on the problem of safety in reinforcement learning. The process of incorporating constraints into the training process of the policy conventionally involves adding safety-related hyperparameters. As a promising direction for reliably deploying RL algorithms in real-world systems, obtaining the optimal hyperparameter values is an important task to ensure that the policy is optimally trained and the constraints are suffieciently satisfied. Hence, the main goal of this research was to propose safe RL methods that are able to achieve competitive performance in both aspects of reward optimality and safety metrics. The main questions of interest studied in this research are:

- How can the safety-related hyperparameters in safe RL methods be meta-optimized so that they are more suitable for real-world deployment?
- Can novel safe RL methods be developed that go beyond the Lagrangian formulation and directly tune the trade-off between optimality and safety?
- How capable are the RL-based method with the safety formulation added to the

policy optimization process to be deployed for training on real-world systems?

- How does the meta-optimized safe RL methods perform compared to the previous safe RL methods in the literature?

This thesis addressed the above questions as follows.

In **Chapter 3** a novel safe RL method, called Meta SAC-Lag, based on constrained MDP was introduced. Using the basic architecture of Soft Actor-Critic (SAC) method, Meta SAC-Lag uses meta-gradient optimization to optimize the safety-related hyperparameters. In each epoch of training, after optimizing the inner parameters (such as the policy network), the constraint threshold is first updated to reflect the adjustment of safety based on the updating process of the inner parameters. Following that, the exploration parameter is meta-optimized to prevent the policy from exploring too unsafely and exploit safer regions. Furthermore, the safety performance of the algorithm was studied in five simulation environments across three main safety topics of locomotion (Humanoid-Velocity environment), obstacle avoidance (Franka CloseDrawer and Car-Circle2 environments), and robotic manipulation (Fetch Push-Topple and Egg Manipulation) to showcase the applicability of the algorithm across a wide range of robotics tasks. Finally, Meta SAC-Lag was deployed and trained on a real-world task of pouring coffee to assess the capability of performing favorably on a robotic system.

In **Chapter 4**, the problem of safe RL was studied in the context of multi-objective policy optimization. Compared with the Lagrangian formulation of safe RL, the multi-objective policy optimization can provide the advantage of performing competitively with lower number of safety-related hyperparameters. This is due to the fact that the multi-objective formulation directly tunes the trade-off between optimality and safety and optimizes both of them, simultaneously. The lower number of safety-related hyperparameters enables the policy to be more robust to suboptimal

hyperparameter tuning and makes it more suitable for general use. To this end, a safety augmentation scheme, called SORL, was proposed which utilized the signals of the safety critic to guide the policy towards safer behaviors. Furthermore, based on some basic assumptions regarding the environment, the safety guarantee condition for the converged policy of SORL was shown. The safety guarantee process introduced an aggressiveness parameter that provided an intuitive way to tune the safety-related hyperparameter of SORL. Moreover, the combination of the aggressiveness parameter and the meta-gradient optimization idea in Chapter 3 provided a constrained meta-optimization approach, called ASORL, for automatically tuning the safety-related hyperparameter of SORL. Finally, the formulation of SORL was evaluated in seven safety-critical tasks across three safety topics of system-level safety (Cheetah-no-flip-Velocity, Ant-Velocity, Hopper-Velocity, Walker2D-Velocity), collision avoidance (Ant-Circle, Drone-Circle), and safe manipulation (Egg Manipulation). Compared with six different state-of-the-art safe RL methods, SORL and ASORL showed better or competitive performance in both aspects of optimality and safety metrics while having fewer safety-related hyperparameters to tune.

5.2 Future Directions

There are many interesting future research directions plausible in the context of safe RL in this research. The list below includes some suggestions for research topics:

i The constraint adjustment and safe exploration methodology proposed in Chapter 3 was proposed under the off-policy SAC architecture. The formulation of this idea under on-policy methods such as TRPO and PPO algorithms and their performance comparison with Meta SAC-Lag can be interesting; in particular, how the two aforementioned safety-related hyperparameters values are

optimized during training process?

ii The safe RL formulation in this research considered constraints as hard constraints and aimed to prevent safety violation throughout a training episode. However, soft constraints or budget-based constraints provide important applications. Hence, the extension of the methods proposed in this research can also be impactful.

iii The safe RL methods proposed in this research aimed at lowering the number of safety-related hyperparameters while potentially speeding up the policy convergence to the safe region. However, another subfield of safe RL methods is the safe exploration where usually a hard threshold is defined and the policy is prevented to choose actions deemed too risky by the algorithm. Future research endeavours can also focus on using the meta-optimization formulation to adjust the safety threshold based on the interaction with the environment.

iv The irrecoverability characterization in Chapter 4 provides a natural and powerful approach for analysing an MDP. An interesting future research direction can include developing methods to identify irrecoverable states and approximate the number of steps until constraint violation. Doing so can result in safe RL methods that can not only be safe in the converged policies but also in the training process.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[3] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[4] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[5] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[6] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io*, 2018.

[7] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

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

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

[10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[11] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[12] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[14] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2019.

[15] Weiye Zhao, Tairan He, Rui Chen, Tianhao Wei, and Changliu Liu. State-wise safe reinforcement learning: A survey. *arXiv preprint arXiv:2302.03122*, 2023.

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

[17] Yun Shen, Michael J. Tobia, Tobias Sommer, and Klaus Obermayer. Risk-sensitive reinforcement learning. *Neural Computation*, 26(7):1298–1328, 2014.

[18] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

[19] Yongshuai Liu, Jiaxin Ding, and Xin Liu. Ipo: Interior-point policy optimization under constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4940–4947, 2020.

[20] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-based constrained policy optimization. *arXiv preprint arXiv:2010.03152*, 2020.

[21] Dan A Calian, Daniel J Mankowitz, Tom Zahavy, Zhongwen Xu, Junhyuk Oh, Nir Levine, and Timothy Mann. Balancing constraints and rewards with meta-gradient d4pg. *arXiv preprint arXiv:2010.06324*, 2020.

[22] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, and Alois Knoll. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*, 2022.

[23] Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *The 30th international joint conference on artificial intelligence (ijcai)*, 2021.

[24] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. Safe learning in robotics: From learning-based

control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):411–444, 2022.

[25] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

[26] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[27] Haonan Yu, Wei Xu, and Haichao Zhang. Towards safe reinforcement learning with a safety editor policy. *Advances in Neural Information Processing Systems*, 35:2608–2621, 2022.

[28] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE conference on decision and control (CDC)*, pages 6059–6066. IEEE, 2018.

[29] Hao-Lun Hsu, Qiuhua Huang, and Sehoon Ha. Improving safety in deep reinforcement learning using unsupervised action planning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5567–5573. IEEE, 2022.

[30] Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future. *Advances in Neural Information Processing Systems*, 34:13859–13869, 2021.

[31] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922, 2021.

[32] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[33] Ahmad Beirami, Meisam Razaviyayn, Shahin Shahrampour, and Vahid Tarokh. On optimal generalizability in parametric learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[34] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International conference on machine learning*, pages 1568–1577. PMLR, 2018.

[35] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[36] Yufei Wang and Tianwei Ni. Meta-sac: Auto-tune the entropy temperature of soft actor-critic via metagradient. *arXiv preprint arXiv:2007.01932*, 2020.

[37] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[40] Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. *Advances in Neural Information Processing Systems*, 36, 2024.

[41] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[42] Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. Conservative safety critics for exploration. *arXiv preprint arXiv:2010.14497*, 2020.

[43] Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023.

[44] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[45] Sven Gronauer. Bullet-safety-gym: A framework for constrained reinforcement learning. 2022.

[46] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[47] Andrew Melnik, Luca Lach, Matthias Plappert, Timo Korthals, Robert Haschke, and Helge Ritter. Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks. *Frontiers in Robotics and AI*, page 57, 2021.

[48] Homayoun Honari, Mehran Ghafarian Tamizi, and Homayoun Najjaran. Safety optimized reinforcement learning via multi-objective policy optimization. *arXiv preprint arXiv:2402.15197*, 2024.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[50] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[51] Amir M. Soufi Enayati, Ram Dershan, Zengjie Zhang, Dean Richert, and Homayoun Najjaran. Facilitating sim-to-real by intrinsic stochasticity of real-time simulation in reinforcement learning for robot manipulation. *IEEE Transactions on Artificial Intelligence*, pages 1–15, 2023.

[52] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[53] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

[54] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148. Citeseer, 2008.

[55] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, pages 11–22. PMLR, 2020.

[56] Abbas Abdolmaleki, Sandy H Huang, Giulia Vezzani, Bobak Shahriari, Jost Tobias Springenberg, Shruti Mishra, Dhruva TB, Arunkumar Byravan, Konstantinos Bousmalis, Andras Gyorgy, et al. On multi-objective policy optimization as a tool for reinforcement learning. *arXiv preprint arXiv:2106.08199*, 2021.

[57] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[58] Dobromir Marinov and Daniel Karapetyan. Hyperparameter optimisation with early termination of poor performers. In *2019 11th Computer Science and Electronic Engineering (CEEC)*, pages 160–163. IEEE, 2019.

[59] Chiara Di Francescomarino, Marlon Dumas, Marco Federici, Chiara Ghidini, Fabrizio Maria Maggi, Williams Rizzi, and Luca Simonetto. Genetic algorithms for hyperparameter optimization in predictive business process monitoring. *Information Systems*, 74:67–83, 2018.

[60] Eitan Altman. *Constrained Markov decision processes*. Routledge, 2021.

[61] Sven Gronauer. Bullet-safety-gym: A framework for constrained reinforcement learning. Technical report, mediaTUM, 2022.

[62] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *Adaptive and Natural Computing Algorithms: 9th International Conference, ICANNGA 2009, Kuopio, Finland, April 23-25, 2009, Revised Selected Papers 9*, pages 380–390. Springer, 2009.

[63] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[64] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 4, 2011.

[65] Yun Shen, Michael J Tobia, Tobias Sommer, and Klaus Obermayer. Risk-sensitive reinforcement learning. *Neural computation*, 26(7):1298–1328, 2014.

[66] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.