Meta SAC-Lag: Towards Deployable Safe Reinforcement Learning via MetaGradient-based Hyperparameter Tuning

Homayoun Honari
hmnhonari@uvic.ca
Department of Mechanical Engineering
University of Victoria

Amir Mehdi Soufi Enayati
amsoufi@uvic.ca
Department of Mechanical Engineering
University of Victoria

Mehran Ghafarian Tamizi
mehranght@uvic.ca
Department of Electrical and Computer Engineering
University of Victoria

Homayoun Najjaran
najjaran@uvic.ca
Department of Mechanical Engineering
Department of Electrical and Computer Engineering
University of Victoria

Abstract

Safe Reinforcement Learning (Safe RL) is one of the prevalently studied subcategories of trial-and-error-based methods with the intention to be deployed on real-world systems. In safe RL, the goal is to maximize reward performance while minimizing constraints, often achieved by setting bounds on constraint functions and utilizing the Lagrangian method. However, deploying Lagrangian-based safe RL in real-world scenarios is challenging due to the necessity of threshold fine-tuning, as imprecise adjustments may lead to suboptimal policy convergence. To mitigate this challenge, we propose a unified Lagrangian-based model-free architecture called *Meta Soft Actor-Critic Lagrangian* (Meta SAC-Lag). Meta SAC-Lag uses meta-gradient optimization to automatically update the safety-related hyperparameters. The proposed method is designed to address safe exploration and threshold adjustment with minimal hyperparameter tuning requirement. In our pipeline, the inner parameters are updated through the conventional formulation and the hyperparameters are adjusted using the meta-objectives which are defined based on the updated parameters. Our results show that the agent can reliably adjust the safety performance due to the relatively fast convergence rate of the safety threshold. We evaluate the performance of Meta SAC-Lag in five simulated environments against Lagrangian baselines, and the results demonstrate its capability to create synergy between parameters, yielding better or competitive results. Furthermore, we conduct a real-world experiment involving a robotic arm tasked with pouring coffee into a cup without spillage. Meta SAC-Lag is successfully trained to execute the task, while minimizing effort constraints. The success of Meta SAC-Lag in performing the experiment is intended to be a step toward practical deployment of safe RL algorithms to learn the control process of safety-critical real-world systems without explicit engineering.

1 Introduction

Reinforcement Learning (RL) is one of the most important paradigms for learning to control physical systems. However, a major shortcoming of RL is its need for exploration and extensive trial and error. For that reason, while we observe its wide success in various domains such as Energy systems Perera & Kamalaruban (2021), Video games Shao et al. (2019), and Robotics Andrychowicz et al. (2020), the real-world deployment of these algorithms to learn the control process poses is challenging Dulac-Arnold et al. (2019) since the exploration process might lead the system to states that might damage the system and incur heavy costs to the user. To this end, safe RL methods aim to address this issue by optimizing the policy such that it is compliant with the

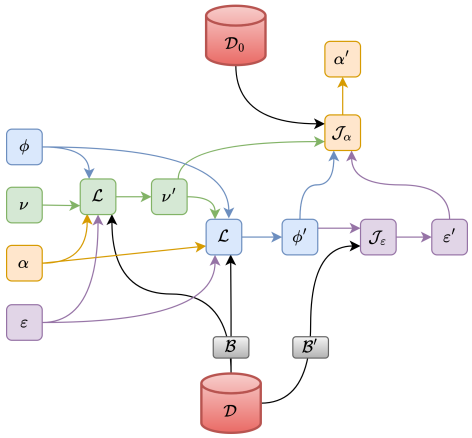


Figure 1: The Computational Graph of the Meta SAC-Lag.

constraints. The constraints are defined such that they aim to prevent the system from exceeding its physical limitations.

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 Gu et al. (2022); Brunke et al. (2022), 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.

To address these challenges, based on the Soft Actor-Critic (SAC) architecture Haarnoja et al. (2018b), our algorithm 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 Haarnoja et al. (2019); Wang & Ni (2020). However, addressing the safety compliance of SAC has been limited. To this end, we propose a threshold-free safety-aware exploration optimization pipeline. In addition, our approach optimizes the safety threshold according to the overall performance of the policy. We are able to update the aforementioned hyperparameters using the metagradients w.r.t. to the meta-objectives which are computed based on the gradients of the internal learnable parameters. Finally, to assess the performance of Meta SAC-Lag, as depicted in Fig. 4, we study its performance against several baseline algorithms in five simulated robotic tasks with four different application themes. We observe that our method attains better or comparable results in terms of safety or reward performance while automatically tuning the safety-related hyperparameters. Furthermore, we present a safety benchmark test case, 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. We deploy and train Meta SAC-Lag in a real-world setup using Kinova Gen3 robot. Our 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.

Our main contributions can be summarized as:

- We propose a Lagrangian-based safe RL method able to automatically adjust the constraint bounds.
- 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 test environment, called *Pour Coffee*, is presented, and, with minimal prior safety-related hyperparameter tuning, Meta SAC-Lag is deployed and trained on a real-world Kinova Gen3 setup. The algorithm successfully achieves the task objective with minimized effort exerted on the robot.

2 Related Work

The constrained Markov decision process (CMDP), is the theoretical building block of safe RL. CMDPs have been widely studied in the RL paradigm Sutton & Barto (2018); Altman (2021) and are solved using Lagrangian methods Bertsekas (2012). In this regard, Shen et al. (2014) devised the risk-sensitive policy optimization (RSPO) algorithm which sequentially decreases the Lagrangian multiplier to zero. Furthermore, Stooke et al. (2020) updates the multiplier using PID control. Additionally, Reward-constrained policy optimization (RCPO) employs dual gradient descent optimization for the policy and Lagrange multiplier Tessler et al. (2018).

In another aspect, metagradient optimization has been explored thoroughly in RL hyperparameter tuning. Initially, model-agnostic meta-learning (MAML) Finn et al. (2017) introduced meta-optimization of initial weights to enable fast task adaptation within a few gradient descent steps. In a different approach, Meta-Gradient RL Xu et al. (2018) extended the concept to learn the hyperparameters of return functions online. This paradigm offered a general approach, applicable to other RL hyperparameters. Subsequently, similar techniques were applied for auto-tuning other RL hyperparameters, such as exploration thresholds Haarnoja et al. (2018a), entropy temperature in SAC Wang & Ni (2020), auxiliary tasks and sub-goals Veeriah et al. (2021), and differentiable hyperparameters of loss functions Zahavy et al. (2020). Despite these advancements, metagradient methods have not been extensively explored in constrained RL paradigms Gu et al. (2022), with few applications focusing on ensuring safety in sensitive learning tasks, as seen in the work by Calian et al. (2020). The authors utilized meta-gradients to update the Lagrange multiplier learning rate in an off-policy RL framework.

The Lagrangian methods are not the sole approach taken toward solving safe RL. Thananjeyan et al. (2021) trained a recovery policy in parallel to the task policy and used it whenever the task policy chose actions deemed too risky. More prominently, model-based RL and safety guarantees for risk aversion during training are proposed. Koppejan & Whiteson (2011) used the neuroevolutionary approach to exploiting domain expertise to learn safe models for model-based RL. Thomas et al. Thomas et al. (2021), in a different approach, used near-future imagination to plan safe trajectories ahead of time. Moldovan & Abbeel (2012) focused on risk aversion in MDPs using near-optimal Chernoff bounds. Lyapunov functions have also been used to guarantee safety during training Berkenkamp et al. (2017); Chow et al. (2018), though constructing Lyapunov functions remains a challenge due to their typically hand-crafted nature and the absence of clear principles for agent safety and performance optimization.

3 Background

In this section, we investigate the background by exploring essential preliminary concepts that serve as the foundation for this paper. We start with the discussion of the CMDP framework. Furthermore, we delve into the formulation of the safety critic and SAC.

3.1 Constrained Markov Decision Process (CMDP)

CMDP comprises the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r, c, \gamma_r, \gamma_c, \rho_0 >$ where S denotes the state space, A represents the action space, and r denotes the reward function: $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$. The transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ defines the likelihood $\mathcal{P}(s'|s, a)$ of moving from state s to s' by executing action a. The probability distribution function $\rho_0 : \mathcal{S} \mapsto [0, 1]$ denotes initial state distribution of the framework. Furthermore, $c(s)$ is the constraint indicator function which determines whether state s violates the constraint functions specified by C: $c(s) = \mathbb{1}[C(s) == 1]$. Parameters $\gamma_r \in [0, 1)$ and $\gamma_c \in [0, 1)$ serve as discount factors for reward and safety critics, respectively. Ultimately, the solution to CMDP is represented by the policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ which is the probability distribution over actions. The value function associated with policy π for a specific

state-action pair (s, a) and the corresponding recursive equation, known as the Bellman equation, can be formulated as follows:

$$Q_r^\pi(s,a) = \mathbb{E}_{s_t\sim\mathcal{P}, a_t\sim\pi}[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)|s_0=s, a_0=a] = \mathbb{E}_{s'\sim\mathcal{P}}[r(s,a)+\gamma V_r^\pi(s')] \tag{1}$$

Additionally, the primary function of the safety critic is to estimate the probability of a policy failure occurring in the future, determined by the expected cumulative discounted probability of failure.

$$Q_c^\pi(s,a) = \mathbb{E}_{s_t\sim\mathcal{P}, a_t\sim\pi}\big[c(s)+(1-c(s))\sum_{t=1}^{\infty}[\gamma_c^t c(s_t)]\big] = \Pr[c(s)==1] + \gamma_c \mathbb{E}_{s'\sim\mathcal{P}}\big[(1-c(s))V_c^\pi(s')\big] \tag{2}$$

Finally, the main objective of an RL algorithm in a CMDP framework is to find a policy to maximize expected return while satisfying the constraints starting from the initial state s_0:

$$\begin{aligned} \pi^* &= \underset{\pi\in\Pi}{\text{argmax}}\ \mathcal{J}_r^\pi = \underset{\pi\in\Pi}{\text{argmax}}\ \mathbb{E}^\pi_{s_0\sim\rho_0}[\sum_{t=0}^{\infty}\gamma^t r_t] \\ \text{s.t. } \mathcal{J}_c^\pi &= \mathbb{E}^\pi_{s_0\sim\rho_0}[\sum_{t=0}^{\infty}\gamma_c^t c_t] \le \varepsilon \end{aligned} \tag{3}$$

3.2 Soft Actor Critic (SAC)

SAC Haarnoja et al. (2018b) optimizes a stochastic policy in an off-policy manner, utilizing two neural networks: one for estimating Q-function (critic) and another for policy updates (actor). A key feature of SAC is entropy regularization, where the policy aims to strike a balance between maximizing expected return and maximizing entropy. This balance mirrors the exploration-exploitation trade-off; higher entropy encourages greater exploration, potentially accelerating learning and prevent convergence to suboptimal solutions.

Considering ω_r and ϕ as parameters representing the critic and actor networks, respectively, training these networks involves sampling a batch of samples from the replay buffer. ω_r is updated by taking the gradient through the mean squared error (MSE) loss between the critic output and the target value:

$$\mathcal{J}_r^{Q_{\omega_r}} = \mathbb{E}_{(s,a,r)\sim\mathcal{D}}[\frac{1}{2}(Q_{\omega_r}(s,a) - Q_r^{\text{tar}}(s,a))^2], \tag{4}$$

where Q_r^{tar} is calculated using Eq. 1:

$$Q_r^{\text{tar}}(s,a) = \mathbb{E}_{s'\sim\mathcal{P}(s,a)a'\sim\pi_\phi}[r(s,a)+\gamma_r(Q_r(s',a') - \alpha log(\pi_\phi(a'|s')))] \tag{5}$$

Furthermore, the policy π_ϕ is optimized by taking the gradient through the critic and the expected entropy of the policy:

$$\mathcal{J}_r^{\pi_\phi} = \mathbb{E}_{\substack{s\sim\mathcal{D}\\ a\sim\pi_\phi}}[\alpha log(\pi_\phi(a|s)) - Q_{\omega_r}(s,a)] \tag{6}$$

Finally, it is important to note that the safety critic (Q_{ω_c}) defined in Section 3.1 is trained using the same loss formulation in Eq. 4, without the entropy term.

4 Method

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

4.1 Metagradient Optimization

Metagradient optimization is the process with which we can optimize the hyperparameters that are not a part of the main loss function. Fundamentally, these meta-parameters[1] dictate the dynamics of the system and

[1] In this paper, we use hyperparameters and meta-parameters terms interchangeably.

direct it toward a certain behavior. In the context of metagradient reinforcement learning Xu et al. (2018), in abstract terms, the learnable system variables are parameterized as θ. These parameters are updated to θ' by following the rule:

$$\theta' = \theta + f(\mathcal{J}, \theta, \eta, \mathcal{B}) \tag{7}$$

where η is the list of hyperparameters, $\mathcal{B}$ a mini-batch of experience, and f the gradient of the objective function $\mathcal{J}$ w.r.t. θ. Furthermore, the optimization process of the meta-parameters η can be formulated based on the updated parameter θ':

$$\eta' = \eta + \beta_\eta \frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \eta} = \eta + \beta_\eta \frac{\partial \mathcal{J}'(\theta', \eta, \mathcal{B}')}{\partial \theta'} \frac{\mathrm{d}\theta'}{\mathrm{d}\eta} \tag{8}$$

where $\mathcal{J}'$ is the meta-objective used for the optimization of the meta-parameters, β_η the learning rate associated with η, and $\mathcal{B}'$ a resampled mini-batch validation set similar to the cross-validation method in the meta-optimization literature Beirami et al. (2017). Finally, $\frac{\mathrm{d}\theta'}{\mathrm{d}\eta}$ can be calculated as:

$$\frac{\mathrm{d}\theta'}{\mathrm{d}\eta} = \left(I + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \theta}\right) \frac{\mathrm{d}\theta}{\mathrm{d}\eta} + \frac{\partial f(\mathcal{J}, \theta, \eta, \mathcal{B})}{\partial \eta} \tag{9}$$

4.2 SAC-Lagrangian

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

$$\begin{aligned} \pi_\phi^* = \max_{\pi_\phi \in \Pi} \mathcal{J}_{r_{\pi_\phi}}^{\pi_\phi} = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi}} [Q_{\omega_r}(s,a) - \alpha \log \pi_\phi(s,a)] \\ \text{s.t. } \mathcal{J}_c^{\pi_\phi} = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi}} [Q_{\omega_c}(s,a)] \leq \varepsilon \end{aligned} \tag{10}$$

Naturally, multiple constraints can be defined for the policy to consider all of them. However, in this paper, in order to keep the formulation simple and general, we consider a single constraint signal that is the result of the superposition of all the constraint functions. In this paper, in contrast to Haarnoja et al. (2018b), we refrain from considering α as an additional constraint and aim to optimize it through metagradient optimization.

Furthermore, the optimization process of policy in Eq. 10 is formulated by casting it as a Lagrangian loss and backpropagating through the loss:

$$\min_{\nu \geq 0} \max_{\pi_\phi \in \Pi} \mathcal{L}(\pi_\phi, \nu, \varepsilon, \alpha) = \mathcal{J}_{r_{\pi_\phi}}^{\pi_\phi} - \nu(\mathcal{J}_c^{\pi_\phi} - \varepsilon) = \mathbb{E}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi}} [Q_{\omega_r}(s,a) - \alpha \log \pi_\phi(s,a) - \nu(Q_{\omega_c}(s,a) - \varepsilon)] \tag{11}$$

where ν is the Lagrange multiplier.

4.3 Meta SAC-Lag

Following the conventional notation in the context of gradient-based hyperparameter optimization Franceschi et al. (2018), we split the parameters into *inner* and *outer* parameters. Rather than a one-shot optimization as in Eq. 7, we propose a sequential updating approach. We define and update the inner parameters as:

$$\theta'_{\text{inner}} = \begin{bmatrix} \nu' \\ \phi' \end{bmatrix} = \begin{bmatrix} \nu \\ \phi \end{bmatrix} + \begin{bmatrix} -\nabla_\nu \mathcal{L}(\pi_\phi, \nu, \varepsilon, \alpha) \\ \nabla_\phi \mathcal{L}(\pi_\phi, \nu', \varepsilon, \alpha) \end{bmatrix} \tag{12}$$

Furthermore, in the same sequential manner, we first update ε and then α:

$$\theta'_{\text{outer}} = \begin{bmatrix} \varepsilon' \\ \alpha' \end{bmatrix} = \begin{bmatrix} \varepsilon \\ \alpha \end{bmatrix} + \begin{bmatrix} \nabla_\varepsilon \mathcal{J}_\varepsilon(\pi_{\phi'}) \\ \nabla_\alpha \mathcal{J}_\alpha(\pi_{\phi'}, \nu', \varepsilon') \end{bmatrix} \tag{13}$$

where $\mathcal{J}_\varepsilon$ and $\mathcal{J}_\alpha$ correspond to the objective functions of ε and α, respectively. To this end, we intended to design the objective function for ε solely based on the performance of the resultant policy. Our 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:

$$\mathcal{J}_\varepsilon(\pi_{\phi'}) = \mathbb{E}_{\substack{s\sim\mathcal{D}\\ a\sim\pi_{\phi'}}}[\nu'_{\text{copy}} Q_{\omega_c}(s,a) - Q_{\omega_r}(s,a)] \tag{14}$$

The objective function $\mathcal{J}_\varepsilon$ is consistent with the objective function of the policy. This is evident by comparing Eq. 14 with the gradient w.r.t. the policy parameters ϕ in Eq. 11. 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. It is important to note that we use ν'_{copy} to indicate that we merely use ν' value in the objective function and not include its gradient w.r.t. ε. We observed better performance by the gradient detachment in our early experiments which may be due to the injection of bias in ν' into its optimization process. Furthermore, to optimize the exploration value α, Wang & Ni (2020) 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, we propose the objective function of α as:

$$\mathcal{J}_\alpha(\pi_{\phi'}, \nu', \varepsilon') = \max_{0<\alpha\leq 1} \mathbb{E}_{\substack{s_0\sim\rho_0\\ a\sim\pi^{\text{det}}_{\phi'}}}[Q_{\omega_r}(s_0,a) - \nu'(Q_{\omega_c}(s_0,a) - \varepsilon')] \tag{15}$$

where $\pi^{\text{det}}_{\phi'}$ indicates the deterministic action value output by the policy. Basically, we use the expectation of the Lagrangian formulation evaluated in the initial states encountered by the agent. The learning process of Meta SAC-Lag is presented in Algorithm 1. Moreover, to gain a better understanding of the gradient relations, illustration of the optimization process of Meta SAC-Lag is depicted in Fig. 1.

5 Experiments

In this section, we evaluate the performance of Meta SAC-Lag. Specifically, our 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?

5.1 Test Benchmarks and Baselines

In order to evaluate the performance of Meta SAC-Lag, we employ five environments across three main safety topics of locomotion, obstacle avoidance, and manipulation. The details and motivation of our choice of environments have been discussed in Appendix C. Furthermore, it is important to note that in the training process, we treat the constraints as hard constraints and terminate the episode whenever a violation has happened in the system. 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 Eq. 11 to optimize the policy.
- **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 Eq. 11.
- **RCPO-MetaSAC**: To showcase the effectiveness of our safe exploration technique, we use the $\hat{Q}(s,a)$ formulation in RCPO and optimize α using the approach proposed in Wang & Ni (2020).
- **Meta SAC-Lag** $\mathcal{J}_{nl}$: Inspired by Honari et al. (2024), we experiment with a nonlinear objective function for ε specified as:

$$\mathcal{J}^{nl}_\varepsilon(\pi_{\phi'}) = \mathbb{E}_{\substack{s\sim\mathcal{D}\\ a\sim\pi_{\phi'}}}\left[\begin{cases} Q_{\omega_r}(s,a)Q_{\omega_c}(s,a) & \text{if } Q_{\omega_r}(s,a) < 0 \\ Q_{\omega_r}(s,a)(1 - Q_{\omega_c}(s,a)) & \text{otherwise} \end{cases}\right] \tag{16}$$

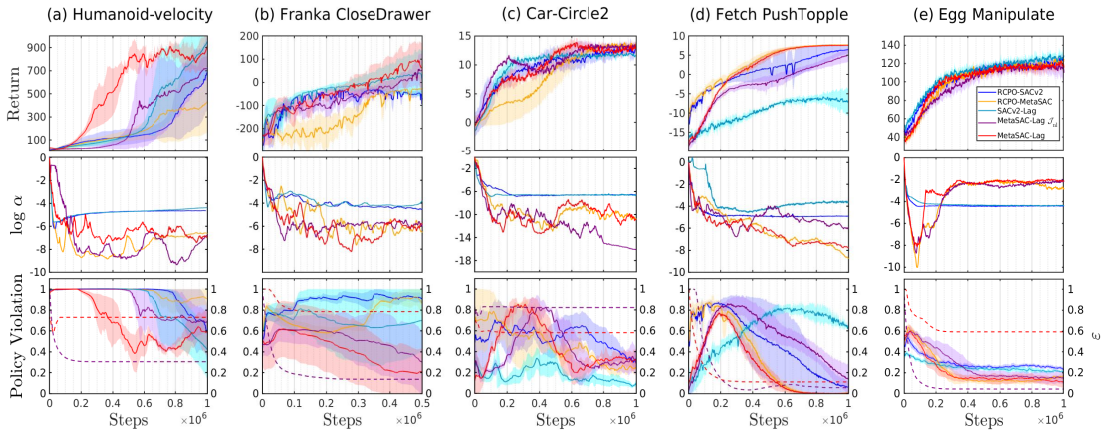


Figure 2: Performance of Meta SAC-Lag compared with the baseline algorithms. **(Top row):** Reward performance during the learning process. (Higher values are better) **(Middle row):** The value of Exploration hyperparameter (α). **(Bottom row):** Episodic policy safety performance of the algorithms during the learning process. (Lower values are better). The dashed lines illustrate the constraint threshold value (ε).

Essentially, $\mathcal{J}_{\varepsilon}^{nl}$ can have the advantage of no reliance on external parameter values, as opposed to Eq. 16 which uses ν' in the objective formulation.

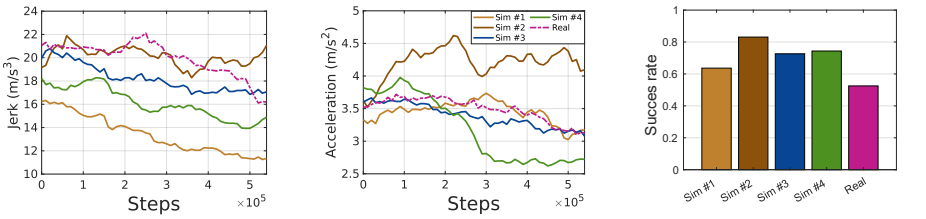
To have a fair comparison, we tune the values of ε and ν for SACv2-Lag and RCPO-SACv2. Also, we use the values of RCPO-SACv2 for RCPO-MetaSAC. The values are outlined in Table 1. For the value of α, SACv2 constrains the policy entropy as $\mathbb{E}_{\substack{s\sim\mathcal{D}\\a\sim\pi}}[-\log(\pi(s,a))] \geq \mathcal{H}$ and defines the α loss as $\min_{\alpha>0}\mathcal{L}(\alpha) = \mathbb{E}_{\substack{s\sim\mathcal{D}\\a\sim\pi}}[\alpha(\log(\pi(s,a)) + \mathcal{H})]$. The authors propose the formula $\mathcal{H} = -\dim(\mathcal{A})$ as their target entropy. Furthermore, two important initial hyperparameter values of Meta SAC-Lag are automatically tuned; therefore, we set $\varepsilon = 1$ and $\alpha = 1$ as their initial values. Moreover, due to their similar training pipelines, we use the initial values of ν for SACv2-Lag in Table 1 for Meta SAC-Lag. We also set $\gamma_r = 0.99$ and γ_c for all the tasks. The results indicate the mean and variance of the performance of the algorithms across multiple independent runs.

5.2 Simulation Results

The simulation results are depicted in Fig. 2. 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 be updated based on the optimally achieved value of ε. Finally, regarding the comparison between Eq. 14 and Eq. 16 we observe consistently better performance of Eq. 14 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.

5.3 Real-World Deployment

Deployability can be regarded as one of the most important obstacles in using RL for learning to control real-world systems Enayati et al. (2023). Choosing unsafe actions might lead the system to states that might



(a) End Effector Jerk (b) End Effector Acceleration (c) Total Success Rate of the Methods

Figure 3: 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.

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 our purpose of assessing the deployability of a safe RL method, we propose a simple, yet important, safe RL testbench. This task, which we call *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 Coumans & Bai (2016–2021). The environment specifics are discussed in Appendix D.

We conduct experiments with Meta SAC-Lag 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 2, three experiments (Simulation #2, #3, #4) utilize different reward shaping schemes along with various constraint definitions. Moreover, we trained Meta SAC-Lag 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 Fig. 3. 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 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.

6 Conclusions

The paper 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, we studied the performance of our algorithm in five simulated embodied applications with the themes of locomotion, obstacle avoidance, robotic manipulation, and dexterous manipulation. We 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, we conducted an experiment in a real-world setup involving a practical coffee-pouring robotic environment without any explicit safety-related reward shaping. We deployed the algorithm on the Kinova Gen3 robot and showed that the proposed algorithm can be helpful for real-world safety-sensitive applications by reducing the reliance on heuristic implementation of safety. We

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.

References

Eitan Altman. *Constrained Markov decision processes*. Routledge, 2021.

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

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

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

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

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

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*, 5:411–444, 2022.

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.

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

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

Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

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*, pp. 1–15, 2023. doi: 10.1109/TAI.2023.3299252.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

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*, pp. 1568–1577. PMLR, 2018.

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.

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*, 2018a.

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*, 2018b.

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.

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

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)*, pp. 5567–5573. IEEE, 2022.

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.

Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary intelligence*, 4:219–241, 2011.

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*, pp. 270–282. PMLR, 2018.

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.

Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

A.T.D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, 2021. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2020.110618. URL https://www.sciencedirect.com/science/article/pii/S1364032120309023.

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

Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

Yun Shen, Michael J. Tobia, Tobias Sommer, and Klaus Obermayer. Risk-sensitive reinforcement learning. *Neural Computation*, 26(7):1298–1328, 2014. doi: 10.1162/NECO_a_00600.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pp. 9133–9143. PMLR, 2020.

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

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

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.

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.

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*, pp. 5026–5033. IEEE, 2012.

Vivek Veeriah, Tom Zahavy, Matteo Hessel, Zhongwen Xu, Junhyuk Oh, Iurii Kemaev, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of options via meta-learned subgoals. *Advances in Neural Information Processing Systems*, 34:29861–29873, 2021.

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

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

Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. A self-tuning actor-critic algorithm. *Advances in neural information processing systems*, 33:20913–20924, 2020.

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.

A Meta SAC-Lag Pseudocode

B Implementation Details

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 $\mathcal{D}_s$ 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. We used a sampled batch $\mathcal{B} \subset \mathcal{D}$ to train the critic networks and the *inner* parameters ν and ϕ. Following that, as discussed in Section 4.1, we use resampled $\mathcal{B}' \subset \mathcal{D}$ and $\mathcal{D}_0$ 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 Beirami et al. (2017); Franceschi et al. (2018). Moreover, following the original architecture Haarnoja et al. (2018b), Meta SAC-Lag uses two critic and safety critic networks to prevent the overestimation of the value functions. To this end, the target values in Eq. 5 are calculated as $\min\{Q_{\bar{\omega}_{r_1}}, Q_{\bar{\omega}_{r_2}}\}$ and $\max\{Q_{\bar{\omega}_{c_1}}, Q_{\bar{\omega}_{c_2}}\}$, respectively. 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 Lillicrap et al. (2015), the target networks aim to increase the stability of the training process and are calculated using the polyak averaging: $\bar{\omega} = \tau\omega + (1-\tau)\bar{\omega}$. The hyperparameter $\tau \in (0, 1)$ typically has a value near zero.

Finally, it is also worth mentioning, in contrast to the original SAC, we use RMSProp Ruder (2016) 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 Wang & Ni (2020).

C Benchmark details

In order to study how the proposed algorithm will perform in safety-critical robotic scenarios, we use 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 controller's actions make the system exceed its limits, e.g., the velocity is higher than a certain threshold or the robot is falling to the

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**
      $\mathcal{D}_s \leftarrow \mathcal{D}_s \cup (s_t, a_t, c_t, s_{t+1})$
    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 $\mathcal{D} \cup \mathcal{D}_s$ (Eq. 2)
    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. 4)
    $\nu' \leftarrow \nu - \beta_\nu \nabla_\nu \mathcal{L}(\pi_\phi, \nu, \varepsilon, \alpha)$ using $\mathcal{B}$ (Eq. 11)
    $\phi' \leftarrow \phi + \beta_\phi \nabla_\phi \mathcal{L}(\pi_\phi, \nu', \varepsilon, \alpha)$ using $\mathcal{B}$ (Eq. 11)
    Resample $\mathcal{B}' = \{s \in \mathcal{D}\}$
    $\varepsilon' \leftarrow \varepsilon + \beta_\varepsilon \nabla_\varepsilon \mathcal{J}_\varepsilon(\pi_{\phi'})$ using $\mathcal{B}'$ (Eq. 14)
    $\alpha' \leftarrow \alpha + \beta_\alpha \nabla_\alpha \mathcal{J}_\alpha(\pi_{\phi'}, \nu', \varepsilon')$ using $\mathcal{D}_0$ (Eq. 15)
    $\nu \leftarrow \nu', \phi \leftarrow \phi', \varepsilon \leftarrow \varepsilon', \alpha \leftarrow \alpha'$
    if $c_t == 1$ **then** Break

Table 1: Hyperparameter values (ε and ν) of the comparison methods

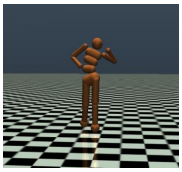
Environment / Parameter	ε	Meta SAC-Lag SACv2-Lag	RCPO-SACv2 RCPO-MetaSAC
Humanoid-Velocity	0.4	10	10
Franka DrawerClose	0.6	10	10
Car-Circle2	0.5	100	1
Fetch PushTopple	0.5	1000	10
Egg Manipulate	0.5	100	1

ground. For that purpose, we use the Mujoco-based Todorov et al. (2012) Humanoid-Velocity environment from the Safety Gymnasium codebase Ji et al. (2024). 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.

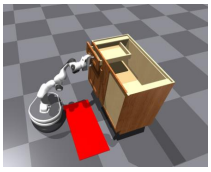
- **Obstacle Avoidance**: In many real-world robotic applications, there are mobile robots with manipulation capabilities. An important constraint of these systems is achieving their goal while avoiding certain regions in their surroundings. We adopt Isaac Gym-based FreightFrankaCloseDrawer Liang et al. (2018). In this setup, the robot attempts to get near a drawer and close it while avoiding a red region. In addition, we use Car-Circle2 task Ji et al. (2024) 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, we use two embodied scenarios. For the **robotic manipulation** task we use Push Topple Bharadhwaj et al. (2020); Hsu et al. (2022) environment where the robotic arm must relo-

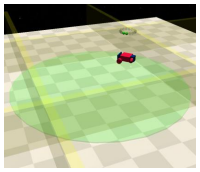
cate a box without toppling it. Furthermore, in the **dexterous manipulation** scenario, we adopt the Egg Manipulate task where the agent must rotate an egg to a specific orientation without dropping it or exerting a force of more than 20 N. For both tasks, we use the Gymnasium Robotics codebase de Lazcano et al. (2023).



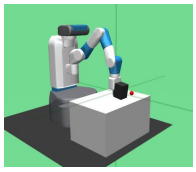
(a) Humanoid-Velocity



(b) Franka CloseDrawer

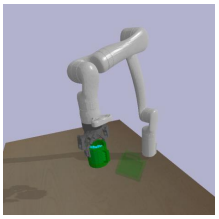


(c) Car-Circle2

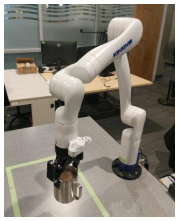


(d) Fetch PushTopple

(e) Egg Manipulate



(f) Pour Coffee (simulation)



(g) Pour Coffee (real setup)

Figure 4: 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.

D Real-world deployment details

We define the state space $\mathcal{S} = \left\{ X_{cup} \cup O_{cup} \cup \dot{X}_{cup} \cup X_{goal} \cup O_{goal} \right\}$ 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, we hierarchically define the reward function for reaching and pouring the coffee based on the Euclidean distance between the cup and the goal $d = ||X_{cup} - X_{goal}||_2$:

$$r(s, a, s') = \begin{cases} r_1 \cdot d + r_2 \cdot ||\ddot{X}_{cup}|| + r_3 \cdot \mathbb{1}[\text{spillage}] & \text{if } d > d_{\text{thresh}} \\ -|\phi_{cup} - \phi_{goal}| + 10 & \text{otherwise} \end{cases} \tag{17}$$

Table 2: *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	✓	✗	✗	✓	✓

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, we can define another constraint 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 Zhu et al. (2020), is the fact that it will eliminate the need to engineer the reward function to minimize the jerk and acceleration of the robot.