The present work was submitted to the Institute for Data Science in Mechanical Engineering.
Diese Arbeit wurde vorgelegt am Institut für Data Science im Maschinenbau.

Learning on Safe Value Functions
Lernen von Sichere Wertfunktionen

Project Report
Master-Projektarbeit

Presented by / Vorgelegt von

Tsung Yuan Tseng
Matr.Nr.: 416164

Supervised by / Betreut von	Alexander von Rohr, M.Sc. (Institute for Data Science in Mechanical Engineering)
1st Examiner / 1. Prüfer	Univ.-Prof. Dr. sc. Sebastian Trimpe (Institute for Data Science in Mechanical Engineering)

Aachen, November 22, 2021

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

[illegible]

Aachen, den November 22, 2021

Abstract

Safety means an agent never visits failure states, e.g., a legged robot never falls to the ground. Safety is of great importance because the failures usually mean painful reset time and physical wear and tear in real-world scenarios. However, current methods to encode safety generally do not scale to high-dimensional tasks or require accurate dynamics. Our goal is to learn the set of all possible safe policies by learning a safe value function, which enables safe exploration for arbitrary tasks. We propose a practical algorithm for learning safe value functions in a model-free manner. We firstly used neural networks to parameterize safe value functions to avoid failure as a safety supervisor, and then we use the learned safety supervisor to guide transfer learning for a new task. We evaluate our algorithm and show that in the general reinforcement learning setting, on the hovership and inverted pendulum tasks, once we learn safe value functions for avoiding failures, learning for new tasks is efficient and the failure rate is significantly lower (0 failure rate on the hovership task and 33 times lower failure rate on the inverted pendulum task) while being able to achieve outstanding performance in the end.

Contents

1 Introduction

Robot learning on hardware often has good promises on performance as there will be no discrepancy between simulations and the real world. In this setting, however, the failures are costly in terms of money, time, or human safety when it is in proximity to humans. Namely, physical damages, manual reset to initialized states, and collisions between human beings and machines are some examples. Aubin *et al.* [1] introduced viability kernel which is the set of the states for agents to avoid failure. Algorithms directly computing the viability kernel are available but have several challenges such as the assumption of accurate dynamics models and they generally do not scale well. These make them hard to use in practice. Alternatively, Heim *et al.* [2] presented the concept of a safety measure based on viability theory which could be learned using Gaussian processes (GPs) and used as a safety function without the viability kernel being explicitly computed. However, some open challenges remain, e.g., scaling to high-dimension tasks and improving sampling efficiency. Massiani *et al.* [3] investigated rigorously the safe value function in the general context of reinforcement learning (RL) and proved that it is optimal with respect to the original task while encoding the safety by adding a sufficient penalty term for failures, but it opens up the question if parameterization still leads to valid safe value functions, which needs further empirical evaluations.

The goal of this project is to extend the theoretical work of [3] to practical RL algorithms and provide empirical evaluations. Specifically, we want to show that we can learn safe value functions and, in consequence, viable sets as defined by [4] and [2], which then will be used to enable safe learning for the new task.

Problem Statement: We consider the standard RL problem on a Markov Decision Process (MDP), which is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, \gamma, r)$. $\mathcal{S}$ and $\mathcal{A}$ are the continuous state and action spaces respectively, T is the transition dynamics, r is the reward function, and γ is the discount factor. Furthermore, we introduce the failure set $\mathcal{S}_F$, a set containing states that considered failed, so as to specify the constraint function:

$$c(s, \pi) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t, s_{t+1}) \sim \rho_\pi} [\delta_{\mathcal{S}_F}(s_{t+1}) \mid s_0 = s], \tag{1.1}$$

where $\delta_{\mathcal{S}_F}$ is the indicator function whose evaluation is one for the state in the failure set $\mathcal{S}_F$; otherwise is zero. Our aim is to learn the policy π that maximizes the following standard RL objective constrained by Eq. 1.1.

$$\underset{\pi}{max} \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \rho_\pi} [r(s_t, a_t, s_{t+1})] \ s.t. \ c(s_t, \pi) = 0, \tag{C}$$

where ρ_π is the marginals of trajectory distribution induced by π.

We then note the information gained from learning the problem (C) in a source domain as:

$$\mathcal{I}_s \sim \mathcal{P}_s, \tag{1.2}$$

and information gained from learning the problem (C) in a target domain as:

$$\mathcal{I}_t \sim \mathcal{P}_t. \tag{1.3}$$

We address (C) in two phases:

1. First, the task of being safe is defined in a source domain, and an agent learns safety information $\mathcal{I}_s$ accordingly.
2. Second, an arbitrary task is defined in a target domain, and an agent learns to behave so that the objective in (C) is maximized by combining the transferred safety knowledge $\mathcal{I}_s$ and target information $\mathcal{I}_t$.

Contributions:

1. We propose a practical algorithm, in which we first learn the task for avoiding failures, and then transfer the safety knowledge to learn a new given task.

2. On hovership example as a proof of concept, we show that once the agent equips with accurate enough safety knowledge, in the second learning stage, the learning is safe and sampling-efficient while preserving optimality compared to learning from scratch.
3. We evaluate our framework on the high-dimensional task, i.e., inverted pendulum, to showcase the benefits of using safety knowledge and shed light on future works.

2 Background

This chapter introduces several fundamental objects. In particular, we first deal with the definition of safety and viability [2], [4]. Secondly, we revisit the theory of safe value functions [3]. Finally, the state-of-the-art soft actor critic algorithm [5] is introduced. Our approach is built upon those three building blocks.

2.1 Safety and viability

Definition 1. *(Viability Kernel) [2]. Viability kernel is a maximal set of all states $s \in \mathcal{S}_V$ in which there exists at least one action $a \in \mathcal{A}$ such that the next state $s' = T(s, a)$ is still inside $\mathcal{S}_V$.*

By its definition, the state outside of viability kernel $\mathcal{S}_V$ will fail in finite time or is already in failure set $\mathcal{S}_F$. We denote this set as the unviability kernel. $\mathcal{S}_U = \mathcal{S} \setminus \mathcal{S}_V$

Definition 2. *(Viable Set) [2]. The viable set $\mathcal{Q}_V \coloneqq \mathcal{S} \times \mathcal{A}$ is a maximal set of all state-action pair $q \in \mathcal{Q}_V$ such that the next state $s' = T(q)$ is still inside $\mathcal{S}_V$.*

Definition 3. *(Trajectory). A trajectory means a sequence of state-action pairs of an agent by following a policy π.*

Definition 4. *(Safety). Safety means an agent's trajectory always stays in the viable set $\mathcal{Q}_V$.*

2.2 Safe value functions

Note that the standard RL problem is to maximize the expected sum of rewards:

$$\underset{\pi}{max} \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \rho_\pi}[r(s_t, a_t, s_{t+1})) \mid s_0 = s]. \tag{U}$$

One could already observe the difference between problem (C) is that it is an unconstrained problem. Now we could define the value function:

$$V^\pi(s) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\rho_\pi}[r(s_t, a_t, s_{t+1})) \mid s_0 = s]. \tag{V}$$

Definition 5. *(Safe Value Function) [3]. Given a value function v defined as V, if all optimal policies that give rise to optimal value function $V^*(s) = \max_\pi V^\pi(s)$ are safe and optimal with respect to problem (C), then v is a safe value function.*

In order to result in a safe value function, one could modify the rewards that only shape the values on the unviability kernel $\mathcal{S}_U$ without affecting the counterpart on viability kernel $\mathcal{S}_V$ such that the optimality is unaffected. In other words, with this, we are making the unviability kernel less attractive than the viability kernel for an agent to visit.

2.3 Soft actor critic

Soft actor critic (SAC) is a state-of-the-art and off-policy algorithm that optimizes the expected sum of rewards and entropy of the policy $\mathcal{H}(\pi(\cdot|s)$ over stochastic policy [5]. Our method uses SAC as a backbone. We define the Q value function:

$$Q^\pi(s,a) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\rho_\pi}[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{T-1} \gamma^t \mathcal{H}(\pi(\cdot|s_t)) \mid s_0 = s, a_0 = a], \tag{2.1}$$

and the Bellman equation for $Q^\pi(s,a)$:

$$Q^\pi(s,a) = \mathbb{E}_{(s',a')\sim\rho_\pi}[r(s,a,s') + \gamma(Q^\pi(s',a') + \alpha\mathcal{H}(\pi(\cdot|s'))]. \tag{2.2}$$

Since Eq. 2.1 is an expectation, $Q^\pi(s,a)$ can be approximated with samples:

$$Q^\pi(s,a) \approx \frac{1}{|\triangle|} \sum_{(s,a,r,s')\in\triangle} [r(s,a,s') + \gamma(Q^\pi(s',\tilde{a}') - \alpha\ log\pi(\tilde{a}'|s'))],\ \tilde{a}' \sim \pi(\cdot|s'), \tag{2.3}$$

where $\triangle$ is a set storing interaction experience.

We approximate the value function with a neural network. We note a Q function approximated by the Q network parameterized by ϕ as:

$$Q_\phi(s,a) \approx Q^\pi(s,a). \tag{2.4}$$

Likewise, the policy function approximated by the policy network parameterized by θ is noted as:

$$\pi_\theta(s) \approx \pi(s). \tag{2.5}$$

Therefore, one could turn the right-hand side of Eq. 2.3 into the target, leading to the loss function:

$$L(\phi) = \frac{1}{|\triangle|} \sum_{(s,a,r,s') \in \triangle} [Q_\phi(s,a) - (r(s,a,s') + \gamma(Q_\phi(s',\tilde{a}') - \alpha\ log \pi_\theta(\tilde{a}'|s'))]^2, \tag{2.6}$$

The Q network updates the weights by minimizing the above loss function. Regarding the policy network, it learns the weights by minimizing the following loss function:

$$L(\theta) = \mathbb{E}_{s \sim \triangle}[D_{KL}(\pi_\theta(\cdot|s)||\frac{exp(Q_\phi(s,\cdot))}{Z_\phi(s)})]. \tag{2.7}$$

Kullback-Leibler Divergence D_{KL} takes two distributions as arguments for measuring how much the difference between these distributions are. By minimizing this loss, intuitively speaking, we move the policy distribution as close as the distribution of the normalized exponential of the Q function.

3 Related Work

Encoding safety in dynamic systems is challenging. One kind of method is to ensure safety by constraining the dynamics states in the viability kernel, which can be computed based on viability theory [1], [6], back-reachable sets [7], [8], or control barrier functions [9]. However, those methods require accurate models and too demanding computation. Model-free learning schemes are then investigated by [10], but is sampling inefficient. Heim and Spröwitz [4] extend viability kernel to viable set; afterwards, Heim *et al.* [2] show that it can be learned using gaussian processes (GPs). However, the computational burden scales cubically with the number of data points because of GPs.

Safe reinforcement learning can be defined as a learning process that optimizes over the policy that maximizes the expected return and respect safety constraints during/after learning [11]. A recent work [12] uses RL schemes to learn safe set, while the policies trained can only stay safe but are not able to accomplish other assigned tasks. Shih *et al.* [13] compute safe sets in a model-based reinforcement learning setup, which mitigates sampling complexity but may have the issue of model bias. In this article, we are solving the safe reinforcement learning problem in a model-free scheme. Our framework can not only allow extracting viable sets but also being able to learn new given tasks under transferable safety knowledge. We alleviate sampling complexity by employing the concept of transfer learning [14]. The recent work [15] uses a similar concept and shows the approach can enable safer, faster, and stable learning on a new task. However, it is still unclear if the framework harms the optimality while encouraging safety. Massiani *et al.* [3] research on safe value functions directly integrates safety into value functions and they show there exists a finite penalty such that penalized value functions are safe value functions. However, some practical aspects remain challenging. For instance, the closed-form solution of minimum but sufficient penalty is often hard to attain [3]. Moreover, as policies and value functions are in general parameterized, there are approximation errors that may break the theory behind. Our work focuses on proposing an algorithm that uses neural networks as function approximators and is able to gradually adapt the penalty.

The focuses of our work are two folds. On the one hand, we aim to develop an algorithm that can scale to high dimensional tasks in order to learn transferable safety information which contains the learned safety-aware policy and the safe value function parameterized by neural networks, as a result, the viable set. On the other hand, by transferring the safety knowledge, the agent in a similar environment is capable of exploring safely and efficiently as well as learning the optimal policy with respect to the new task.

4 Methods

4.1 Penalty auto-tune

We consider the entropy-regularized RL objective as followed:

$$G(s,\pi) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\rho_\pi}[r(s_t,a_t,s_{t+1}) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{4.1}$$

where the temperature parameter α determines the trade-off between reward r and entropy of the policy $\mathcal{H}(\pi(\cdot|s_t))$.

The problem that we want to solve, as defined in (C) but replaced objective with (4.1), is a constrained optimization problem that is hard to solve in general. A common method is to change the constrained optimization problem into an unconstrained optimization problem by adding a penalty term to the objective to penalize the constraint violation:

$$\underset{\pi}{max}\ G(s,\pi) - p \cdot c(s,\pi). \tag{P}$$

Massiani *et al.* [3] prove that, under some mild assumptions, there exists p^* such that for all $p > p^*$, all maximizers of the penalized problem (P) are optimal for the constrained problem (C). Noted that p^* will be different from the proving result in [3] since our expected sum of rewards G includes the entropy term. This makes it hard to tune manually in our formulation. Inspired by the Lagrangian dual problem. We can define the Lagrangian expression for our constraint problem as:

$$\mathcal{L}(\pi,\lambda) = G(s,\pi) - \lambda \cdot c(s,\pi), \tag{4.2}$$

where p is the Lagrangian multiplier λ.

The dual problem is:

$$\underset{\lambda}{min}\ g(\lambda) = \underset{\lambda}{min}\ \underset{\pi}{sup}\ \mathcal{L}(\pi, \lambda). \tag{4.3}$$

Since we cannot directly solve the dual sub-problem, we resort to iteratively solving the dual sub-problem and dual problem using gradient descent. For the readers who are interested in the Langrangian dual problem, Goodwin *et al.* [16] have a good introduction.

As a result, we propose an approach to auto-tune the penalty:

$$\underset{p}{min}\ \underset{\pi}{max}\ G(s, \pi) - p \cdot c(s, \pi). \tag{4.4}$$

We could rewrite Eq. 4.4 as follows:

$$\underset{p}{min}\ \underset{\pi}{max} \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \rho_\pi} [r(s_t, a_t, s_{t+1}) - p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) + \alpha \mathcal{H}(\pi(\cdot|s_t))], \tag{4.5}$$

which iteratively solves maximization and minimization problems. As solving the maximization problem, on the one hand, we apply SAC algorithm by replacing $r(s, a, s')$ defined in Chapter 2.3 with $R(s, a, s')$:

$$R(s, a, s') = r(s, a, s') - p \cdot \delta_{\mathcal{S}_F}(s'). \tag{4.6}$$

While solving the minimization problem, since the optimizer has no control over the first term $G(s, \pi)$ (the decision variable is p), it will just try to increase the value of p such that the second term of the objective is reduced. We then approximate the second term as the following loss function:

$$L(p) = \frac{1}{|\triangle|} \sum_{(s,a,r,s') \in \triangle} -p \cdot \delta_{\mathcal{S}_F}(s'), \tag{4.7}$$

which can be interpreted as an estimation of the failure rate. As time goes on, as the Q function and policy concurrently optimized, when the Q function values are more attractive on Q_V than on Q_U and the policy improves by minimizing Eq. 2.7, that means the constraint is less likely to be violated $\forall s \in S_V$. In the limit, by the time when Eq. 4.7 is close to 0, the penalty p converges. This is a highly versatile framework. For example, one may modify the learning rate for updating p or only update it if the current p is not sufficient based on some criteria.

4.2 Learning safety supervisor

Our framework contains two learning stages, learning safety supervisor and transfer learning with learned safety supervisor. The reason for this proposed method is that there may be a budget limitation that the developers can only afford failures for one round but they are asked to learn different tasks in a similar environment. In this scenario, our framework is highly relevant. Learning safety supervisor has a goal to learn the transferable safety knowledge, including the viable set, safe value function, and safe policy, such that a new agent can learn arbitrary tasks safely on top of the information. Below, we detail how to learn the safety information. The viable set Q_V is of great importance because it tells the agent if it is safe or not given the state-action pair. Once we learn accurate Q_V, we could guarantee safety during learning the new task. Hence, we introduce the theory-back method to learn a viable set.

Theorem 1 (Extracting viable set)**.** *If we design $r = 0$, there exists p^* such that for all $p > p^*$, the viable set Q_V can be recovered via:*

$$Q_V = \{q, Q^\pi(q) > 0\}.$$

Proof. With $R(s, a, s') = -p \cdot \delta_{\mathcal{S}_F}(s')$, the value function becomes:

$$Q^\pi(s,a) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\rho_\pi}[\sum_{t=0}^{T-1} -\gamma^t p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) + \alpha \sum_{t=1}^{T-1} \gamma^t \mathcal{H}(\pi(\cdot|s_t)) \mid s_0 = s, a_0 = a]. \tag{4.8}$$

There exists p^*, $\forall q = (s, \pi(\cdot|s)) \in Q_U$:

$$p^* = \frac{\sum_{t=1}^{T-1} \gamma^t \cdot \alpha \mathcal{H}(\pi(\cdot|s_t)}{\gamma^{T-1}} < p \tag{4.9}$$

$$\iff \sum_{t=1}^{T-1} \gamma^t \cdot \alpha \mathcal{H}(\pi(\cdot|s_t) < \sum_{t=0}^{T-1} \gamma^t \cdot p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) \tag{4.10}$$

$$\iff \sum_{t=0}^{T-1} [-\gamma^t p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) + \alpha \sum_{t=1}^{T-1} \gamma^t \mathcal{H}(\pi(\cdot|s_t)] < 0, \tag{4.11}$$

such that $Q^\pi(q) < 0$. Since the entropy term in the expectation is always positive, we directly know $\forall q = (s, \pi(\cdot|s)) \in Q_V$, $Q^\pi(q) > 0$. As a result, the viable set Q_V can be recovered via:

$$Q_V = \{q, Q^\pi(q) > 0\}. \tag{4.12}$$

■

By defining a source domain on MDP: $(\mathcal{S}_s, \mathcal{A}_s, T_s, \gamma_s, r_s = 0)$, we solve this using the proposed algorithm in Chapter 4.1. The agent samples the action according to the current policy and observes the next state by executing the action in the environment. Further on, the experience of every step is stored in the buffer used to update the actor, critic networks, and the penalty. After training, the learned safety supervisor will have information about viable set $\hat{Q}_V$, critic network $\hat{Q}_\phi$, and actor network $\hat{\pi}_\theta$ (Algorithm 1).

4.3 Transfer learning with learned safety supervisor

The core idea in this section is that by transferring the safety knowledge from the first stage to the second stage: transfer learning with learned safety supervisor, we want to perform transfer learning in a target domain on MDP: $(\mathcal{S}_t, \mathcal{A}_t, T_t, \gamma_t, r_t)$ with the guarantee of safety during exploration (Algorithm 2). Note that $\mathcal{S}_t, \mathcal{A}_t, T_t$ are the same as the counterparts specified in the source domain and that the problem is solved by the proposed algorithm in Chapter 4.1. We omit the subscript s and t if the meaning is clear in the context. We transfer the safety knowledge by three measures: (1) Initializing the value function by $\hat{Q}_\phi$. (2) Initializing the policy by

$\hat{\pi}_\theta$. (3) The queriable $\hat{Q}_V$. Particularly, the learned safety supervisor contains not only the pre-trained critic $\hat{Q}_\phi$ but also actor network $\hat{\pi}_\theta$, which conceptually are safe value function for only avoiding failure and the safe policy to give safe actions. These networks are thus perfect candidates as the initialized networks in this transfer learning stage. In addition, during the interaction, the queriable $\hat{Q}_V$ comes into play, i.e., the agent could make inquiries from $\hat{Q}_V$ about whether the current state-action pair is safe as the agent interacts with the environment. As there must be approximation errors in $\hat{Q}_V$, there are chances that the safety action doesn't exist given the current state. In this situation, we do a virtual step of the possibly unsafe action and assign the next state as a failure state closest to the current state without executing the unsafe action in the environment. Further on, we gather the experience and update the networks accordingly. Due to the fact that the agent already knows the safety knowledge and will not waste time exploring unsafe (valueless) states, we expect that transfer learning with learned safety supervisor is safe, more sampling efficient, and enjoys the optimality compared to learning the tasks from scratch.

4.4 Algorithm summary

The exploration behavior hampers safety in RL [17]. A typical RL procedure (Figure 4.1) is to learn the optimal policy for the specific task by interacting with the environment, which may possibly be unsafe during the exploration. Our framework (Figure 4.2) concatenates two blocks but changes the task of the first block into learning safety. The second stage of learning can utilize the safety knowledge to learn the new task.

Algorithm 2 summarizes the whole pipeline, i.e., first to perform safety supervisor learning (Algorithm 1) and secondly transfer learning. This framework is not limited to only perform transfer learning for one time. As long as the safety supervisor of target dynamics is learned, it becomes safe and efficient to do transfer learning whenever a new task is assigned. In conclusion, instead of directly tackling the problem (C), our approach, transfer learning with safety supervisor, address the problem (P) with penalty auto-tune in two learning phases. For the comparison, Algorithm 3 is the method that directly solves the problem (P) with penalty auto-tune but without the pre-trained safety supervisor. To ease the notations, we note the Algorithm 1 as learning safety supervisor (LSS), the Algorithm 2 as transfer

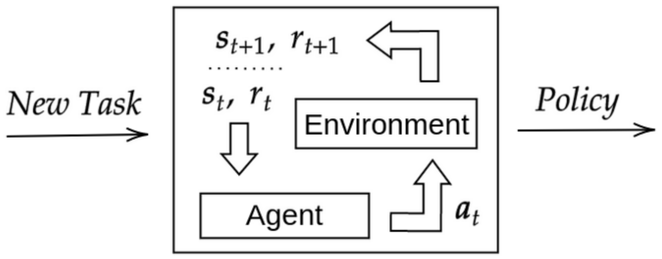

Figure 4.1: A Typical RL procedure. A new task is assigned to the agent and an RL algorithm is applied where the agent and the environment interact with each other. During the interaction, possible unsafe actions may be executed, leading to failed situations.

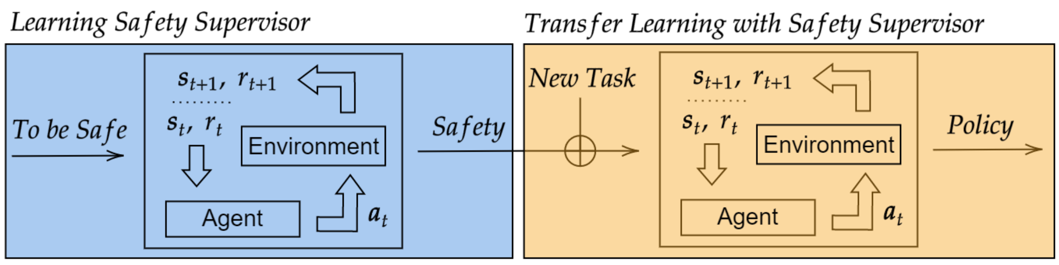

Figure 4.2: Our proposed framework contains (1) learning safety supervisor and (2) transfer learning with safety supervisor. The goal in the first stage is to learn transferable safety knowledge to be used in the second stage. Transfer learning with safety supervisor aims to learn the new task safely with the help of transferable safety knowledge.

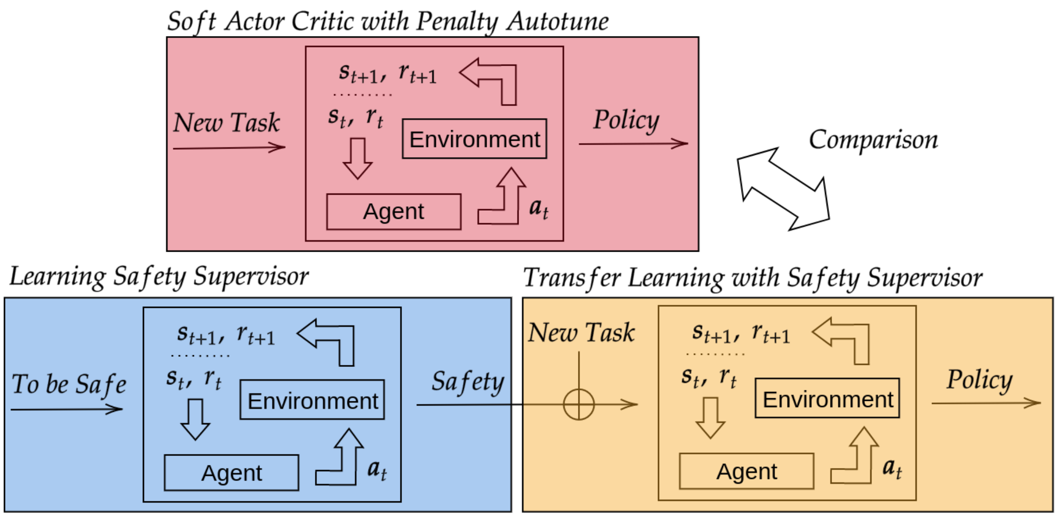

Figure 4.3: Comparison between our proposed framework and learning from scratch. The bottom part of the plot is our framework. The upper part of the plot denotes the method of learning from scratch, soft actor critic with penalty autotune. We compare these two to answer what the benefits of using the learned safety supervisor are.

learning with safety supervisor (TL-SS), and the Algorithm 3 as soft actor critic with penalty auto-tune (SAC-PA).

Algorithm 1 Learning Safety Supervisor

```
1: procedure SAFETY SUPERVISOR
2:     Initialize networks $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
3:     for each interaction steps do
4:         $a_t \sim \hat{\pi}_\theta(a_t|s_t)$
5:         $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$
6:         $\triangle \leftarrow \triangle \cup (s_t, a_t, r_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$    ▷ Assign r=0 everywhere
7:         for each gradient steps do    ▷ If it is time to update
8:             Apply SAC update for $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
9:             $p \leftarrow p - \lambda\hat{\nabla}L(p)$    ▷ Penalty autotune
10:        end for
11:    end for
12:    $\hat{Q}_V = \{q, min(\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}) > 0\}$    ▷ Extract viable set
13:    return $\hat{Q}_V, \hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
14: end procedure
```

Algorithm 2 Transfer Learning with Safety Supervisor

```
1: procedure TRANSFER LEARNING
2:     $Q_V, Q_{\phi_1}, Q_{\phi_2}, \pi_\theta \leftarrow$ Learning Safety Supervisor    ▷ Algorithm 1
3:     for each interaction steps do
4:         $a_t \sim \pi_\theta(a_t|s_t) \forall a \ s.t. \ (s_t, a) \in Q_V$    ▷ Only execute the safe actions
5:         if $\nexists a \ s.t. \ (s_t, a) \in Q_V$ then    ▷ If there is no safe action
6:             $a_t \sim \pi_\theta(a_t|s_t)$    ▷ Choose the action anyway
7:             $s_{t+1} \leftarrow \forall s_{closest} \in \mathcal{S_F}$    ▷ Virtual step
8:         else
9:             $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$    ▷ Safe execution
10:        end if
11:        $\triangle \leftarrow \triangle \cup (s_t, a_t, R_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$    ▷ Designed r for the new task
12:        for each gradient steps do    ▷ If it is time to update
13:            Apply SAC update for $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$
14:            $p \leftarrow p - \lambda\hat{\nabla}L(p)$    ▷ Penalty autotune
15:        end for
16:    end for
17: end procedure
```

Algorithm 3 SAC with Penalty Autotune

```
 1: procedure SAC Penalty Autotune
 2:     Initialize networks $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$        ▷ Without Safety Supervisor
 3:     for each interaction steps do
 4:         $a_t \sim \pi_\theta(a_t|s_t)$
 5:         $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$
 6:         $\triangle \leftarrow \triangle \cup (s_t, a_t, R_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$        ▷ Designed r for desired task
 7:         for each gradient steps do        ▷ If it is time to update
 8:             Apply SAC update for $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$
 9:             $p \leftarrow p - \lambda \hat{\nabla} L(p)$        ▷ Penalty autotune
10:         end for
11:     end for
12: end procedure
```

5 Results

We have tested our algorithm on two dynamics: the hovership spaceship and inverted pendulum to represent a low-dimensional task and a high-dimensional task, which are modified from these repositories [1] [2]. The hovership example, a low dimensional task as a proof-of-concept, is to showcase the validity of our propose framework. The hovership has a 1-D state space representing the height of the machine and a 1-D action space denoting the trust to control the vertical height (Table 5.1). The mission of the hovership is to explore safely the planet which has a dangerous gravitation field. On the other hand, the inverted pendulum has a 4-D state space describing the status of the cart and pendulum; additionally, it has a 1-D action space to move the cart linearly either to the left or right (Table 5.2). For such a high dimensional task, typically the ground truth of the viable set attained by brute force is computationally demanding and often assumed unknown. Below we first detail the experimental set-up for the hovership and inverted pendulum tasks. We then introduce the evaluation metrics along with the experiments we made and finally report and discuss the experiment results.

State/Action	**Variable**	**Min**	**Max**
s	Hovership Height: h	0	2
a	Hovership Thrust: u	0	0.8

Table 5.1: The state-action space of hovership.

Hovership. The hovership spaceship has a thruster against gravity which is stronger while approaching the ground. If it hits the ground, it counts as a failure. When the thruster is fired such that the next state is over the maximum height, then it stays

[1] `https://github.com/sheim/vibly/blob/master/models/hovership.py`

[2] `https://github.com/0xangelo/gym-cartpole-swingup/blob/master/gym_cartpole_swingup/envs/cartpole_swingup.py`

State/Action	**Variable**	**Min**	**Max**
s_1	Cart Position: x	-2.4	2.4
s_2	Cart Velocity: $\dot{x}$	$-\inf$	inf
s_3	Cosine of Pole Angle: $cos\theta$	-1	1
s_4	Sine of Pole Angle: $sin\theta$	-1	1
s_5	Angular Velocity of Pole: $\dot{\theta}$	$-\inf$	inf
a	Force on Cart: f	-1	1

Table 5.2: The state-action space of invented pendulum.

at the highest level. The reward r is designed to be 0, $\forall h$ and $p = 10$ in learning safety supervisor (Algorithm 1). For Algorithm 2 and 3, the desired task is to learn the optimal policy such that the hovership moves from the initial state $h = 1.8$ to the goal state $h = 1.3$ as fast as possible. Therefore, we define $\gamma = 0.8$, $p = 50$, and the following reward function:

$$r = \begin{cases} 50, \ if \ h = 1.3 \pm 0.01 \\ 0, \ otherwise \end{cases} . \tag{5.1}$$

Inverted Pendulum. The inverted pendulum has a motor that can move the cart linearly on the track. The failure states are defined as the cart position x reaches the boundaries ± 2.4 or the pole angle θ exceeds the limits $\pm 45°$. We design $r = 0$ once again for all states and $p = 30$ in Algorithm 1. Regarding Algorithm 2 and 3, the assigned task is to train the agent moving from the initial state $x = 0, \dot{x} = 0, \theta = 0, and \ \dot{\theta} = 0$ to the goal state $x = 1.5$ while stabilizing the pole. Hence, we design the reward function as followed:

$$r = \begin{cases} \frac{1}{2}(1 + cos\theta) + \frac{2}{3}x, \ if \ 0 \leq x \leq 1.5 \\ \frac{1}{2}(1 + cos\theta) - 2x + 4, \ if \ 1.5 < x \leq 2 \\ \frac{1}{2}(1 + cos\theta), \ otherwise \end{cases} . \tag{5.2}$$

Figure 5.1 is the plot for reward functions and failure states.

Evaluation Metric.

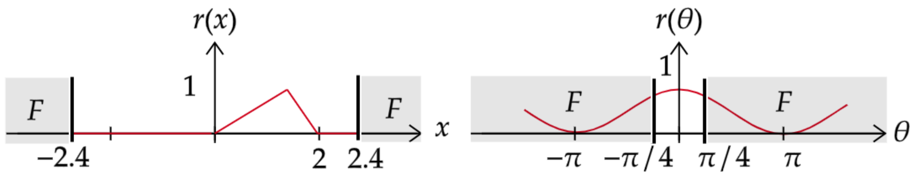

Figure 5.1: Reward functions for inverted pendulum. The reward function of the cart position is shown on the left and the reward function of the pole orientation is shown on the right. The failure states are colored in gray.

- **Root Mean Square Error (RMSE)**. On the hovership example, we quantify the accuracy of extracted viable set $\hat{Q}_V$ by root mean square error:

$$RMSE = \frac{1}{N} \sum_{(s,a)\in Grid} \sqrt{(Q_V(s,a) - \hat{Q}_V(s,a))^2}, \tag{5.3}$$

  where (s,a) is a discretized state-action pair in the region of interest: *Grid*, N describes the total number of pairs, and Q_V is the ground truth computed by brute force.

- **Empty Set Rate**. In theory, if we constraint the agent in viable set $\hat{Q}_V$ exactly the same as ground-truth Q_V, then the agent will always stay in viability kernel S_V. However, since we approximate $\hat{Q}_V$ by neural networks, there always are approximation errors. Thus, with this metric, we would like to know the answer that how often the safety supervisor returns an empty safe action set given fixed evaluation steps.

- **Misjudgment Rate**. We start the agent from the initial state, and execute uniform-randomly the safe action given the state at each time step. The next state of the agent ends up either being stay in viability kernel or not. We record the rate that $\hat{Q}_V$ thinks the action is safe but unsafe in reality. This is an underestimated value because we realize the action is unsafe only after the agent visits the failure set.

- **Failure Rate**. As we want to measure how effective the learned safety supervisor guiding transfer learning without failures is, we monitor the failure rate for given interaction steps during transfer learning.

5.1 Hovership

We test our framework on the hovership for 50 runs. In the first learning stage, learning safety supervisor, we monitor the RMSE between the extracted viable set $\hat{Q}_V$ and ground-truth Q_V computed by brute force (see Figure 1). The RMSE converges to the ground truth. Regarding the second learning phase, transfer learning with safety supervisor, from the Figure 5.2, one can observe our algorithm TL-SS converges faster; thus, it is sampling more efficient than SAC-PA. In the end, the learning performance of TL-SS reaches the same level as SAC-PA. In Table 5.3, the evaluation metrics are summarized. Even though the empty set rate of LSS is not perfect, it still has a perfect failure rate of 0 due to good enough understanding of safety knowledge. It is also because if there is no safe action, we do a virtual step and reset the environment in the next iteration. All in all, the results suggest that the safety supervisor guarantees safety during the second stage of learning and does not affect the agent's final learning performance.

Algorithm	**Failure Rate**	**Empty Set Rate**	**Misjudgement Rate**
TL-SS	0.0000 ± 0.0000		
SAC-PA	0.0008 ± 0.0001		
LSS		0.0028 ± 0.0530	0.0000 ± 0.0000

Table 5.3: This table summarizes the failure rate during learning for TL-SS and SAC-PA where TL-SS manages to stay safe. It also summarizes the empty set rate and misjudgment rate for LSS.

5.2 Inverted Pendulum

We test our algorithm on the inverted pendulum for 10 runs. Since we are dealing with a 6 dimensional state-action space, the ground truth of Q_V is computationally demanding and therefore assumed unknown. We summarize two metrics, empty set rate and misjudgment rate (Table 5.4), to evaluate the learning outcome of $\hat{Q}_V$. We observe that the quantity of empty set rate is worse than misjudgment rate, meaning that the agent doesn't have the overall understanding of safety knowledge so empty set rate is high, and it tries to be conservative when it thinks there is no safe action to be executed so the misjudgment rate is low. For the second phase of

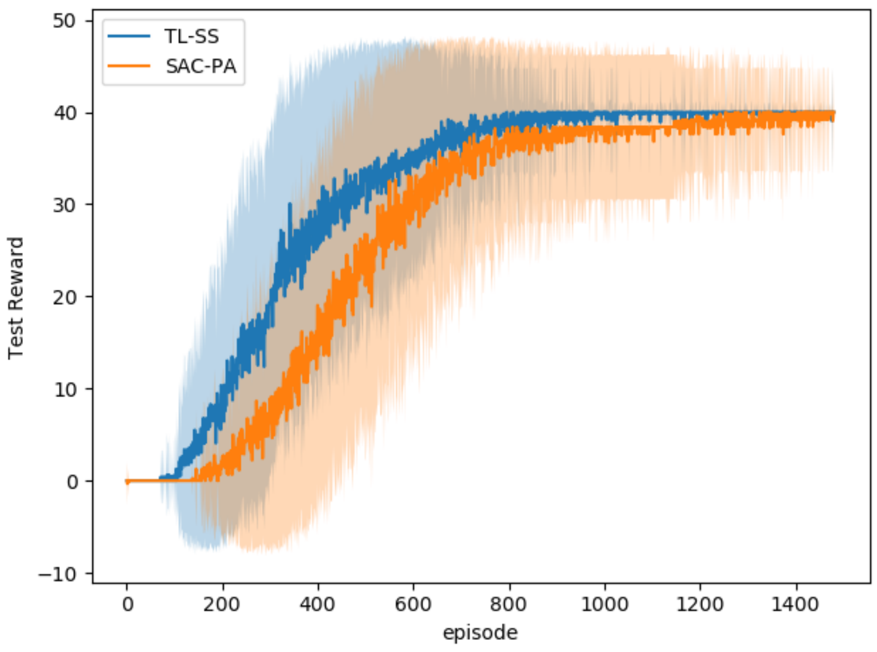

Figure 5.2: Test reward in Algorithm 2, transfer learning with safety supervisor (TL-SS) and Algorithm 3, soft actor critic with penalty auto-tune (SAC-PA) on hovership. The result shows that out method converges faster and have comparable final learning outcome.

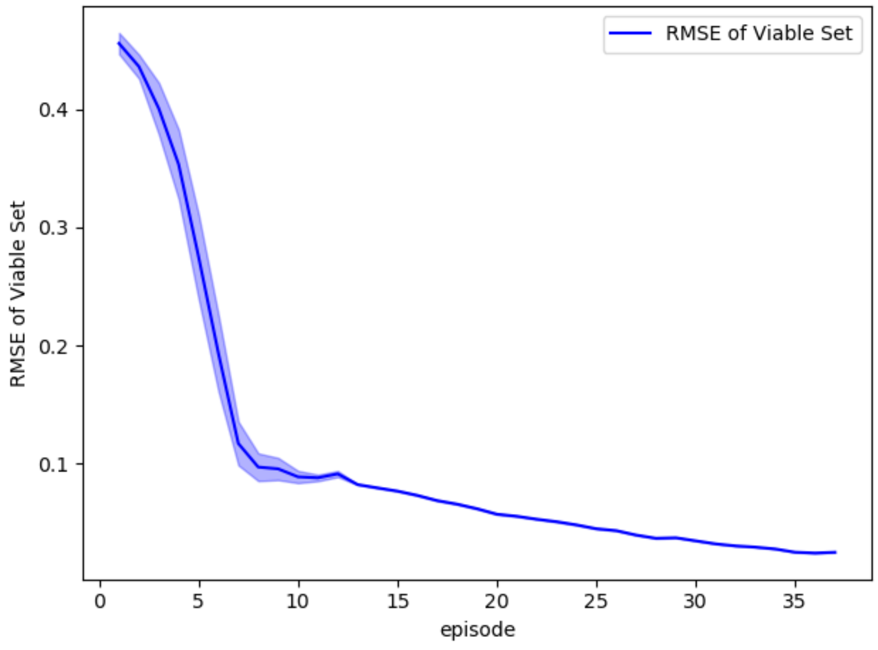

Figure 5.3: RMSE of Viable Set in learning safety supervisor (LSS). The learning curve shows that our method can successfully extract viable set.

learning, TL-SS final learning performance and learning speed outperform SAC-PA (see Figure 5.4) while the failure rate is 33 times lower (Table 5.4).

Algorithm	**Failure Rate**	**Empty Set Rate**	**Misjudgement Rate**
TL-SS	0.0312 ± 0.0554		
SAC-PA	0.6996 ± 0.0554		
LSS		0.0222 ± 0.1473	0.0004 ± 0.0001

Table 5.4: TL-SS has a 33 times lower failure rate compared to SAC-PA, which highlights the advantages of using the learned safety supervisor although the empty set rate and misjudgment rate for LSS are not perfect.

5.3 Discussion

Based on the results of these two experiments, we can observe that if we could learn accurate enough $\hat{Q}_V$ (in the case of hovership example), the safety supervisor indeed can guide the learning safely and efficiently while also achieving outstanding learning outcome. On the inverted pendulum task, TL-SS starts to fail when the agent tries to accelerate to the right and samples the state-action pairs that are out-of-distribution for $\hat{Q}_V$, meaning that the safety supervisor seldom/never visits those samples before. Hence, the accuracy of $\hat{Q}_V$ is the essence of success for avoiding failures. Additionally, we find that the soft actor critic algorithm is not sufficient to explore the high dimensional state-action space, and consequently, learn $\hat{Q}_V$ since some regions in the state-action space are hard to visit.

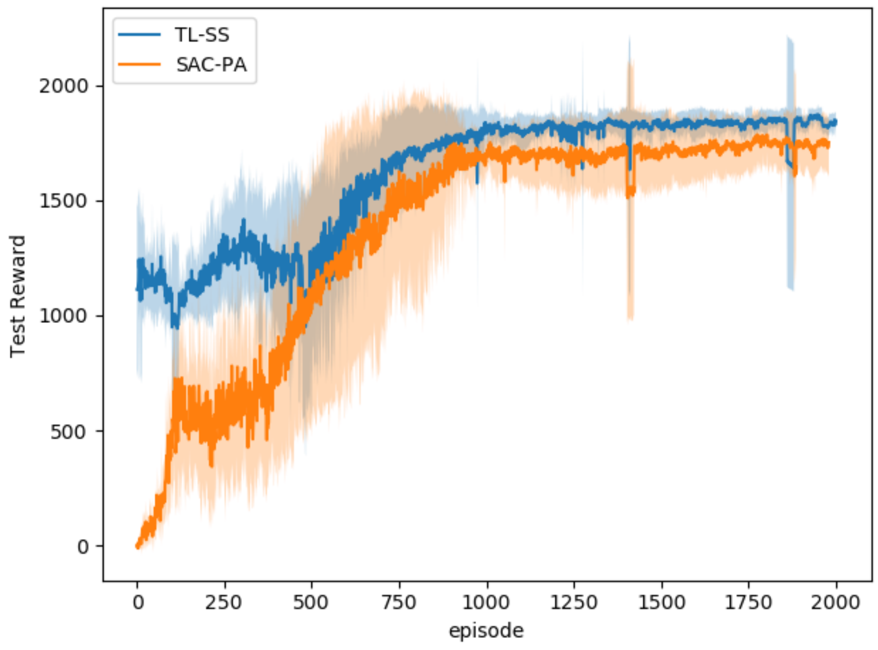

Figure 5.4: Test reward in Algorithm 2, transfer learning with safety supervisor (TL-SS) and Algorithm 3, soft actor critic with penalty auto-tune (SAC-PA) on inverted pendulum. The result shows that our method is more efficient and have better performance in the end.

6 Conclusion

We propose a practical algorithm, in which we first learn the safe value functions for avoiding failures (safety supervisor), and then use learned critic and actor networks as an initialization accompanied with extracted viable set transferred to learn a new given task, both with the possibility of penalty auto-tune. We prove our concept on hovership example that once we learn an accurate enough viable set, in the transfer learning stage, the learning is safe and sampling efficiently compared to learning from scratch. We evaluate our framework on the high-dimensional task, i.e., inverted pendulum, showing that our algorithm reduces the failure rate significantly while having better performance in the end. Active learning on $\hat{Q}_V$ may be one of the future works as it can help the agent be excited to visit the area of the state-action space that is uncertain.

List of Figures

List of Tables

Bibliography

[1] J.-P. Aubin, A. M. Bayen, and P. Saint-Pierre, *Viability theory: new directions.* Springer Science & Business Media, 2011.

[2] S. Heim, A. Rohr, S. Trimpe, and A. Badri-Spröwitz, “A learnable safety measure,” in *Conference on Robot Learning*, 2020, pp. 627–639.

[3] P.-F. Massiani, S. Heim, F. Solowjow, and S. Trimpe, “Safe value functions,” *arXiv preprint arXiv:2105.12204*, 2021.

[4] S. Heim and A. Spröwitz, “Beyond basins of attraction: Quantifying robustness of natural dynamics,” *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 939–952, 2019.

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

[6] A. Liniger and J. Lygeros, “Real-time control for autonomous racing based on viability theory,” *IEEE Transactions on Control Systems Technology*, vol. 27, no. 2, pp. 464–478, 2017.

[7] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, “Hamilton-jacobi reachability: A brief overview and recent advances,” in *2017 IEEE 56th Annual Conference on Decision and Control*, IEEE, 2017, pp. 2242–2253.

[8] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, “A general safety framework for learning-based control in uncertain robotic systems,” *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2018.

[9] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control barrier functions: Theory and applications,” in *2019 18th European control conference*, IEEE, 2019, pp. 3420–3431.

[10] A. Robey, H. Hu, L. Lindemann, *et al.*, “Learning control barrier functions from expert demonstrations,” in *2020 59th IEEE Conference on Decision and Control*, IEEE, 2020, pp. 3717–3724.

[11] J. Garcıa and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[12] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, “Bridging hamilton-jacobi safety analysis and reinforcement learning,” in *2019 International Conference on Robotics and Automation*, IEEE, 2019, pp. 8550–8556.

[13] J. C. Shih, F. Meier, and A. Rai, “A framework for online updates to safe sets for uncertain dynamics,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2020, pp. 5994–6001.

[14] Z. Zhu, K. Lin, and J. Zhou, “Transfer learning in deep reinforcement learning: A survey,” *arXiv preprint arXiv:2009.07888*, 2020.

[15] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, “Learning to be safe: Deep rl with a safety critic,” *arXiv preprint arXiv:2010.14603*, 2020.

[16] G. Goodwin, M. M. Seron, and J. A. De Doná, *Constrained control and estimation: an optimisation approach.* Springer Science & Business Media, 2006.

[17] L. Brunke, M. Greeff, A. W. Hall, *et al.*, “Safe learning in robotics: From learning-based control to safe reinforcement learning,” *arXiv preprint arXiv:2108.06266*, 2021.