Enforcing Hard State-Dependent Action Bounds on Deep Reinforcement Learning Policies

Bram De Cooman[1(✉)][0000−0003−4843−3342], Johan Suykens[1][0000−0002−8846−6352], and Andreas Ortseifen[2][0000−0002−2555−4515]

[1] KU Leuven, ESAT - STADIUS, Leuven, Belgium
{bram.decooman,johan.suykens}@esat.kuleuven.be
[2] Ford Research & Innovation Center, Aachen, Germany
aortseif@ford.com

Abstract. Imposing hard constraints on deep reinforcement learning policies trained with model-free methods is a challenging task. In this paper we specifically focus on constraining the policy's actions, by imposing state-dependent action bounds. Such bounds allow the designer to incorporate prior domain knowledge into the model-free learning framework and can be used to improve the stability or safety of the learned policies. The approach is applied to two benchmark environments and a more complicated autonomous driving problem. When correctly applied, the state-dependent action bounds can provide strong safety guarantees, as well as improve the convergence speed.

Keywords: Reinforcement Learning · Hard Constraints · Action Bounds · Safe Control · Autonomous Driving.

1 Introduction

An advantage of model-free reinforcement learning (RL) is that no model knowledge is required and policies can be trained purely through interactions with an unknown environment. This, however, comes with a cost, as the lack of a model makes these methods less sample efficient than their model-based counterparts. Moreover, it is harder to constrain policies to certain (safe) regions in state-action space without the guidance of a model. In this paper we propose a novel way to embed prior domain knowledge into the model-free learning scheme using state-dependent bounds on the agent's actions. When correctly applied, these bounds can improve sample efficiency as well as provide strong safety guarantees, while keeping the benefits of the model-free learning setting.

1.1 Motivation

The potential benefits of state-dependent action bounds can be illustrated by a virtual driver example in which the agent needs to learn a highway-driving policy — this setup is also analyzed in more depth in Section 4. Obviously, if the agent starts with zero domain knowledge, many collisions will occur early on

in the training process. Incorporating domain knowledge by providing common rules of traffic to the agent helps preventing obvious mistakes (Figure 1.a). As the resulting state-dependent action bounds are effectively hard constraints, it might improve the trust of the general public in the learned policies. Beside the safety aspect, these bounds also speed up training, by constraining the agent to only explore the relevant, safe regions of state-action space. Finally, such bounds can also be used to ensure compliance with traffic regulations required for vehicle homologation in certain markets, such as keeping a minimum safety distance during automated lane changes (Figure 1.b).

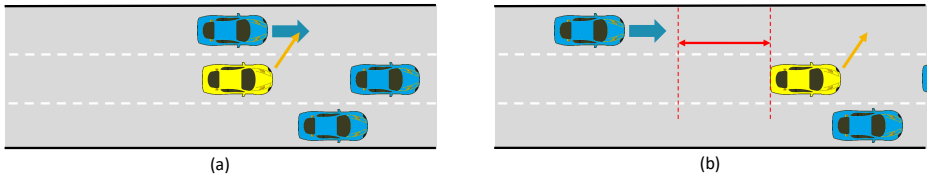

Fig. 1. Schematic overview of two situations on a highway. On the left a situation where an unconstrained virtual driver (yellow) might decide to move towards the occupied left lane. State-dependent action bounds can prevent the virtual driver from executing such hazardous manoeuvres. On the right a minimum safety distance for an automated lane change is visualized, which can be ensured by state-dependent action bounds.

1.2 Main Contributions and Related Work

Safe reinforcement learning is an active field of research for which an extensive overview is given by Garcia et al. [13]. Some approaches consider the setting in which safety must be learned through environmental interactions, which means safety constraints may be violated during training [7, 25]. Other methods try to incorporate the safety constraints *throughout* the learning phase, ensuring exploration itself is also safe [3, 19, 27]. In this paper we follow the latter approach and enforce the state-dependent bounds both during training and deployment.

Closely related to the safe RL domain are methods solving Constrained Markov Decision Processes (CMDP), which beside optimizing for performance (long-term accumulated reward) also take an arbitrary amount of policy constraints into account [1, 7, 25]. Our approach could be considered a special case under this CMDP framework, as we specifically focus on state-dependent action bounds, i.e. upper and lower bound constraints on actions only. One benefit of restricting us to this specific subclass of problems is that we can guarantee hard constraint satisfaction, while the methods solving the more general (and difficult) CMDP problem typically only have near-constraint satisfaction.

For discrete action spaces a commonly used approach is to prune the available action set, such that unsafe actions are excluded. The Deep Constrained Q-learning method [16] can be used to learn such policies. The pruning can either

be done in a preprocessing step, by providing the agent with a safe set of actions to choose from; or in a postprocessing step, after the agent has already made a decision [2]. In the latter case, the agent assigns a list of priorities to each discrete action from which the *safe* action with highest priority is chosen in the pruning step [15, 21]. While our approach is only applicable to continuous action spaces, it could be classified as *preemptive shielding*: the agent is only allowed to select safe actions that lie within the predefined range of state-dependent bounds.

Postposed shielding approaches for continuous action spaces are provided by Dalal et al. [8] and Cheng et al. [6]. In both setups the control pipeline is extended with a "safety controller", which maps potentially unsafe actions selected by the agent to the closest safe actions. This requires an optimization problem to be solved at each timestep, which may not be feasible if computing resources are limited. To alleviate this, Dalal et al. [8] suggest solving the optimization problem analytically, under the extra assumption that only a single constraint is active at the same time.

Chandak et al. [5] study the setting where the action space is stochastic. Such a setting is more generic than ours, as the range of available actions (defined through the state-dependent bounds) is considered to be deterministic and defined before learning starts. Finally, it should be mentioned that many environments make use of *static* action bounds, for example to model actuator saturation. Most standard RL methods are able to deal with that, for instance through using bounded activation functions in the last layer of the actor network. However, such bounds are constant for all states. Hence, to the best of our knowledge, our work is the first to apply *state-dependent* action bounds.

Enforcing such hard, state-dependent bounds on policy actions is the central contribution of this work. These bounds not only prevent exploration and operation in irrelevant parts of the state-action space, but also allow the designer to embed prior domain knowledge in the model-free learning setting; ultimately leading to policies with an improved performance, sample efficiency and safety.

1.3 Organization

This paper is organized as follows. The necessary reinforcement learning essentials are first described in Section 2, followed by an in-depth introduction of state-dependent action bounds in Section 3. Our approach is then evaluated on an autonomous driving problem in the experiments of Section 4. Extra background information and additional experiments can be found in the appendices.

2 Reinforcement Learning

Formally, the Reinforcement Learning (RL) problem can be described by a Markov Decision Process (MDP), written down as the tuple $(\mathcal{S}, \mathcal{A}, \iota, \tau, r, \gamma)$ with $\mathcal{S} \subset \mathbb{R}^S$ denoting the state space, $\mathcal{A} \subset \mathbb{R}^A$ the action space, $\iota(\boldsymbol{s}_0) : \mathcal{S} \to [0\,;1]$ the initial-state distribution, $\tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0\,;1]$ the state transition

model, $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward model and $\gamma \in [0\,;1)$ the discount factor. The state transition model $\tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ describes the probability of going to state $\boldsymbol{s}_{t+1}$ in the next time step, when selecting action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ at the current time step. The reward model $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ provides the agent with a scalar reward for transitioning towards state $\boldsymbol{s}_{t+1}$ when selecting action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$. It is this reward signal that provides the agent with feedback, which can be used to improve its policy $\pi(\boldsymbol{a}|\boldsymbol{s}) : \mathcal{A} \times \mathcal{S} \to [0\,;1]$. This policy describes the probability of taking action $\boldsymbol{a}$ when the agent perceives state $\boldsymbol{s}$. For deterministic policies we will also use the notation $\boldsymbol{a} = \pi(\boldsymbol{s})$.

The goal of the agent is to maximize the future discounted return, given by

$$R_t = \sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}),$$

for all possible initial states and subsequent visited state-action pairs by following its policy π. The optimal policy π^* is thus found as

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi,\tau,\iota}[R_0], \tag{1}$$

where the expectation $\mathbb{E}_{\pi,\tau,\iota}$ is taken over the probability distribution of actions $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$, induced by the policy, and over the probability distribution of states $\boldsymbol{s}_0 \sim \iota(\cdot)$ and $\boldsymbol{s}_{t+1} \sim \tau(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$, induced by the environment.

Instead of working directly with the objective (1), it is often useful to consider the *state-value* function $V_\pi(\boldsymbol{s}) = \mathbb{E}_{\pi,\tau}[R_t \mid \boldsymbol{s}_t = \boldsymbol{s}]$ and *action-value* function $Q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,\tau}[R_t \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}]$. Both of these value functions can also be defined recursively through a relationship known as the Bellman equation

$$\begin{aligned} V_\pi(\boldsymbol{s}) &= \mathbb{E}_{\pi,\tau}[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma V_\pi(\boldsymbol{s}_{t+1}) \mid \boldsymbol{s}_t = \boldsymbol{s}], \\ Q_\pi(\boldsymbol{s}, \boldsymbol{a}) &= \mathbb{E}_{\pi,\tau}[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma Q_\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}]. \end{aligned}$$

More background information regarding the fundamentals of reinforcement learning can be found in the book by Sutton & Barto [24].

There are many different RL algorithms to find an estimate of the optimal policy or value function for the previously described MDP (see [28] for a taxonomy of different approaches). Enforcing state-dependent action bounds on the policies, requires the usage of RL methods with explicit actor networks (policy-based or actor-critic), supporting both continuous state and action spaces. In this work, we specifically focus on two model-free, off-policy, actor-critic methods, which are briefly introduced in the following two subsections. Being actor-critic methods, both an actor network $\mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$, modelling the policy, and a critic network $Q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_Q)$, estimating the optimal state-action value function, are jointly trained. Extra target networks (denoted by primes on the weight vectors $\boldsymbol{\theta}'_i$) are also introduced to improve the stability of the learning process and are updated using Polyak averaging. As the environment dynamics ι, τ remain unknown to the agent in the model-free setting, the agent is required to explore the state-action space during training. The *behavioural policy* $\beta(\boldsymbol{a}|\boldsymbol{s})$ is used for

this exploration, collecting experience tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ which are stored in a replay buffer $\mathcal{B}$. The actor and critic networks are then trained using batches of experience tuples from this replay buffer.

2.1 Deterministic Methods

Two popular deterministic, actor-critic methods are 'Deep Deterministic Policy Gradient' (DDPG) [20] and 'Twin Delayed DDPG' (TD3) [12]. In these methods, the actor network represents the deterministic policy $\boldsymbol{a} = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$. To ensure sufficient exploration of the state-action space, an external source of stochasticity is necessary during training, as a deterministic policy would always take the same actions in the same states. Hence, the behavioural policy $\beta(\boldsymbol{s}) = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu) + \boldsymbol{\epsilon}$ with exploration noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_{expl} I)$ is used to collect experience during training.

The critic network is updated by minimizing a squared temporal difference error

$$L_Q(\boldsymbol{\theta}_Q) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}) \sim \mathcal{B}} \Big[\big(Q(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\theta}_Q) - y_t \big)^2 \Big],$$
$$y_t = r_t + \gamma Q(\boldsymbol{s}_{t+1}, \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}'_\mu); \boldsymbol{\theta}'_Q).$$

The actor network is updated by minimizing $L_\mu(\boldsymbol{\theta}_\mu) = -\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{B}}[Q(\boldsymbol{s}_t, \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu); \boldsymbol{\theta}_Q)]$, resulting in an approximation of the deterministic policy gradient [23]

$$\nabla_{\boldsymbol{\theta}_\mu} L_\mu \approx \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{B}} \Big[\nabla_{\boldsymbol{\theta}_\mu} \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)|_{\boldsymbol{s}=\boldsymbol{s}_t} \nabla_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_Q)|_{\boldsymbol{s}=\boldsymbol{s}_t, \boldsymbol{a}=\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)} \Big].$$

2.2 Stochastic Methods

The 'Soft Actor-Critic' (SAC) [14] method is an example of a stochastic, actor-critic algorithm. In this method, the actor network provides the parameters for the probability distribution used to model the stochastic policy $\pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$. Typically a Gaussian distribution is used, taking $\boldsymbol{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \mathrm{diag}[\boldsymbol{\sigma}_t])$ with $\big[\boldsymbol{\mu}_t; \boldsymbol{\sigma}_t\big] = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$. To ensure a sufficient exploration of the state-action space, this method does not only optimize for long-term reward accumulation, but also tries to maximize the policy's entropy $\pi^* = \arg\max_\pi \mathbb{E}_{\pi, \tau, \iota}[R_0 + \alpha H_0]$, with $H_t = \sum_{k=0}^{\infty} \gamma^k \mathcal{H}[\pi(\cdot | \boldsymbol{s}_{t+k})]$ the future discounted entropy. The critic network estimates the optimal soft Q-function, which takes this extra entropy term into account. The resulting critic loss can then be written down as the soft Bellman residual

$$L_Q(\boldsymbol{\theta}_Q) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}) \sim \mathcal{B}} \Big[\big(Q(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\theta}_Q) - y_t \big)^2 \Big],$$
$$y_t = r_t + \gamma \mathbb{E}_{\boldsymbol{a}_{t+1} \sim \pi}[Q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; \boldsymbol{\theta}'_Q) - \alpha \log[\pi(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)]].$$

To improve the policy, the Kullback-Leibler divergence between the policy and the exponential of the soft Q-function is minimized, leading to the actor loss

$$L_\mu(\boldsymbol{\theta}_\mu) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{B}} \Big\{ \mathbb{E}_{\boldsymbol{a}_t \sim \pi} \big[\alpha \log[\pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}_\mu)] - Q(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\theta}_Q) \big] \Big\}. \tag{2}$$

Finally, the temperature parameter α can be tuned either manually or automatically adapted throughout training, see Haarnoja et al. [14] for further details.

3 Proposed Methodology

Two modifications to the usual reinforcement learning (RL) control pipeline are necessary to enforce state-dependent bounds on policy actions, as shown in Figure 2. First of all, the behavioural policy β and learned policy π are constrained such that their sampled *normalized actions* $\tilde{\boldsymbol{a}}$ always lie within a predetermined, *fixed* interval. In a second step, these normalized actions are rescaled using a chosen mapping σ and the *state-dependent* action bounds $a_{\mathrm{L}}(\boldsymbol{s})$ and $a_{\mathrm{U}}(\boldsymbol{s})$ — encoding the prior domain knowledge to be embedded. The resulting rescaled actions $\boldsymbol{a}$ are then executed in the environment, leading to a next state in the following timestep.

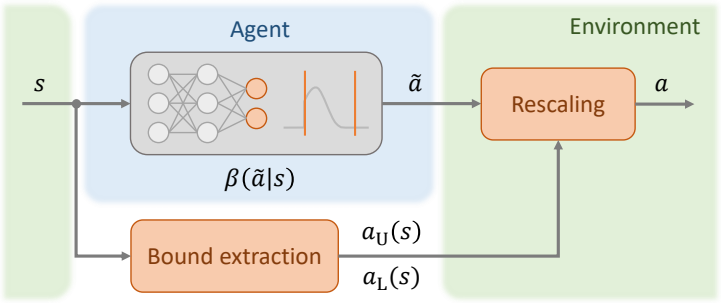

Fig. 2. Schematic overview of the reinforcement learning control pipeline with enforced state-dependent action bounds.

The chosen reinforcement learning algorithm is applied using the normalized actions $\tilde{\boldsymbol{a}}$ instead of the rescaled actions $\boldsymbol{a}$, i.e. $\tilde{\boldsymbol{a}}$ is stored in the replay buffer $\mathcal{B}$ and used for the calculation of the actor (and critic) losses. As a result, from the agent's perspective, the rescaling operation is part of the environment. In case the mapping σ is differentiable with respect to the normalized action inputs, it would be possible to work directly with the rescaled actions, making the rescaling operation part of the agent instead. To simplify the comparison of different (possibly non-differentiable) rescaling functions, we do not investigate such a setup in this paper and leave this for future work.

3.1 Normalized Actions

The specific implementation of policies with bounded normalized actions depends on the chosen RL method used for policy optimization. In essence, it consists of the following two steps. First, squash the outputs of the actor network representing the mean actions $\boldsymbol{\mu}$ to the fixed interval $[\tilde{a}_{\mathrm{L}}\,;\tilde{a}_{\mathrm{U}}]$ using a bounded activation function, such as tanh, in the last layer of the network. Secondly, to make sure no actions outside this fixed interval are sampled, use a bounded

probability distribution, such as the truncated Gaussian distribution $\mathcal{N}_{\tilde{a}_\text{L}}^{\tilde{a}_\text{U}}(\boldsymbol{\mu}, \sigma I)$ with finite support $[\tilde{a}_\text{L}\,; \tilde{a}_\text{U}]$ [4].

Below, two implementations are presented for a state-of-the-art deterministic and stochastic actor-critic method respectively. Note that while we stick to the tanh activation function with bounds $\tilde{a}_\text{L} = -1$, $\tilde{a}_\text{U} = 1$ in this paper, any other bounded and differentiable activation function can be used instead.

TD3 (Deterministic) In deterministic actor-critic methods, such as TD3, the output of the actor network corresponds to the deterministic action to be taken for a given state $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$. To properly explore the state-action space during training, an external source of stochasticity is used, typically a spherical Gaussian with mean $\boldsymbol{a}_t$ and a decaying variance σ: $\boldsymbol{a}_t^\epsilon \sim \mathcal{N}(\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu), \sigma I)$.

Bounding the mean action outputs on the actor network, thus corresponds to using tanh activation in the last layer of the network, resulting in normalized action outputs $\tilde{\boldsymbol{a}}_t$. To ensure satisfaction of the bounds during exploration, the truncated Gaussian distribution is used.

$$\begin{aligned} \pi &: \tilde{\boldsymbol{a}}_t = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu) \\ \beta &: \tilde{\boldsymbol{a}}_t^\epsilon \sim \mathcal{N}_{-1}^{1}(\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu), \sigma I) \end{aligned}$$

SAC (Stochastic) In stochastic actor-critic methods, such as SAC, the output of the actor network corresponds to the parameters of the distribution from which stochastic actions can be sampled. Using a Gaussian distribution for example, we have $\left[\boldsymbol{\mu}_t; \boldsymbol{\sigma}_t\right] = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$ and $\boldsymbol{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \text{diag}[\boldsymbol{\sigma}_t])$.

In this case it suffices to use a tanh activation in the last layer of the μ-head of the actor network, leading to normalized mean action outputs $\tilde{\boldsymbol{\mu}}_t$. Once again, the truncated Gaussian distribution is used to ensure satisfaction of the bounds for all sampled actions. During evaluation, the mode of the learned stochastic policy β is used to retrieve a deterministic policy π.

$$\begin{aligned} \pi &: \tilde{\boldsymbol{a}}_t = \tilde{\boldsymbol{\mu}}_t \\ \beta &: \tilde{\boldsymbol{a}}_t^\epsilon \sim \mathcal{N}_{-1}^{1}(\tilde{\boldsymbol{\mu}}_t, \text{diag}[\boldsymbol{\sigma}_t]) \end{aligned} \qquad \begin{bmatrix} \tilde{\boldsymbol{\mu}}_t \\ \boldsymbol{\sigma}_t \end{bmatrix} = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$$

The SAC method requires the calculation of differentiable log probabilities for the sampled actions (2). For the standard Gaussian distribution, such gradients can be calculated straightforwardly using the *reparametrization trick* [18]. Such a reparametrization trick can not be applied, however, when using the truncated Gaussian distribution. Fortunately, differentiable log probabilities can still be calculated using implicit reparametrization gradients [11].

Note that in Appendix C of the soft-actor critic paper [14] an alternative way of bounding the actor network's outputs is described. The "tanh trick" uses samples from the regular Gaussian distribution which are afterwards passed through the tanh function, squashing each sample to the bounded interval $[-1\,;$ $1]$. While this is also a viable approach, we work with the truncated Gaussian distribution in this paper for consistency with the deterministic methods.

3.2 State-Dependent Rescaling

The sampled normalized actions are afterwards rescaled to the interval $[a_{\text{L}}(\boldsymbol{s})\,; a_{\text{U}}(\boldsymbol{s})]$ determined by the predefined, state-dependent action bounds a_{L} and a_{U}. Different rescaling functions $\sigma(\tilde{a}; \boldsymbol{s}) : [\tilde{a}_{\text{L}}\,; \tilde{a}_{\text{U}}] \times \mathcal{S} \to [a_{\text{L}}(\boldsymbol{s})\,; a_{\text{U}}(\boldsymbol{s})]$ can be used for that purpose. We specifically focus on rescaling functions that are monotonically increasing bijections, to guarantee the structure of the underlying Markov Decision Process (MDP) is not modified (see Subsection 3.3). In this subsection we introduce three such mappings. For simplicity, we assume scalar (normalized) actions, but this can be easily generalized to vectors by applying the σ functions elementwise.

The most straightforward way to rescale a variable is using the linear function

$$\sigma_{\text{lin}}(\tilde{a}; \boldsymbol{s}) = a_{\text{L}}(\boldsymbol{s}) + \frac{a_{\text{U}}(\boldsymbol{s}) - a_{\text{L}}(\boldsymbol{s})}{\tilde{a}_{\text{U}} - \tilde{a}_{\text{L}}}(\tilde{a} - \tilde{a}_{\text{L}}).$$

The benefit of this mapping is its generality, as it is able to handle any range of bounds $[a_{\text{L}}(\boldsymbol{s})\,; a_{\text{U}}(\boldsymbol{s})]$. A drawback is that there is no fixed *anchor point* within the bounds, more precisely there is no $\tilde{a}_0 \in [\tilde{a}_{\text{L}}\,; \tilde{a}_{\text{U}}]$ that is mapped to a fixed a_0. For some control problems such a fixed anchor point can be useful or provide extra (stability) guarantees.[3] The following piecewise linear rescaling function allows to specify such an anchor point

$$\sigma_{\text{pwl}}(\tilde{a}; \boldsymbol{s}) = \begin{cases} a_0 + \dfrac{a_{\text{L}}(\boldsymbol{s}) - a_0}{\tilde{a}_{\text{L}} - \tilde{a}_0}(\tilde{a} - \tilde{a}_0) & \tilde{a} < \tilde{a}_0 \\ a_0 + \dfrac{a_{\text{U}}(\boldsymbol{s}) - a_0}{\tilde{a}_{\text{U}} - \tilde{a}_0}(\tilde{a} - \tilde{a}_0) & \tilde{a} \geq \tilde{a}_0 \end{cases}.$$

Remark that a_0 is fixed and hence $a_{\text{L}}(\boldsymbol{s}) \leq a_0 \leq a_{\text{U}}(\boldsymbol{s})$ should be satisfied for all $\boldsymbol{s}$, making this mapping slightly more restrictive than the previous one. A drawback of this rescaling function is that it is not differentiable at $\tilde{a} = \tilde{a}_0$. Hence, a smoother variant is given by the hyperbolic interpolation function

$$\begin{aligned} \sigma_{\text{hyp}}(\tilde{a}; \boldsymbol{s}) &= a_0 + \frac{p(\boldsymbol{s})(\tilde{a} - \tilde{a}_0)}{q(\boldsymbol{s})(\tilde{a} - \tilde{a}_0) + r(\boldsymbol{s})}, \\ p(\boldsymbol{s}) &= (a_{\text{L}}(\boldsymbol{s}) - a_0)(a_{\text{U}}(\boldsymbol{s}) - a_0)(\tilde{a}_{\text{U}} - \tilde{a}_{\text{L}}), \\ q(\boldsymbol{s}) &= (a_{\text{L}}(\boldsymbol{s}) - a_0)(\tilde{a}_{\text{U}} - \tilde{a}_0) - (a_{\text{U}}(\boldsymbol{s}) - a_0)(\tilde{a}_{\text{L}} - \tilde{a}_0), \\ r(\boldsymbol{s}) &= (\tilde{a}_{\text{L}} - \tilde{a}_0)(\tilde{a}_{\text{U}} - \tilde{a}_0)(a_{\text{U}}(\boldsymbol{s}) - a_{\text{L}}(\boldsymbol{s})). \end{aligned}$$

Figure 3 shows the different rescaling functions for $\tilde{a}_{\text{L}} = -1$, $\tilde{a}_0 = 0$, $\tilde{a}_{\text{U}} = 1$, $a_{\text{L}} = -2$, $a_0 = 0$ and $a_{\text{U}} = 5$. The inverse mapping $\sigma^{-1}(\cdot; \boldsymbol{s})$, from actions a to normalized actions $\tilde{a}$, can be easily calculated for any of the presented rescaling functions by swapping the normalized bounds and anchor $\tilde{a}_{\text{L}}$, $\tilde{a}_0$, $\tilde{a}_{\text{U}}$ with the rescaled bounds and anchor a_{L}, a_0, a_{U}. A proof for this property is provided in Appendix A.

[3] For example, if actions in a local neighbourhood of a_0 stabilize the system around an equilibrium point $\boldsymbol{s}^*$ it might be desirable to map $\tilde{a}_0 = \mathbb{E}[\pi(\cdot|\boldsymbol{s}^*)]$ to a_0.

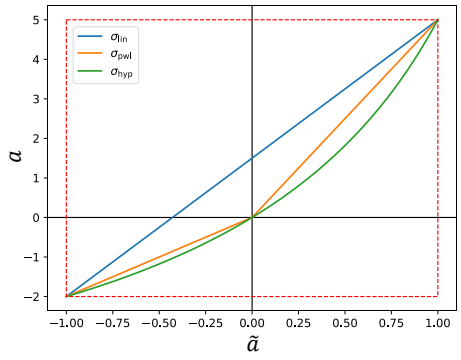

Fig. 3. Comparison of the different rescaling functions used throughout this paper. In this plot the bounds were set to $\tilde{a}_{\text{L}} = -1$, $\tilde{a}_0 = 0$, $\tilde{a}_{\text{U}} = 1$, $a_{\text{L}} = -2$, $a_0 = 0$ and $a_{\text{U}} = 5$.

3.3 Preservation of MDP

The environment dynamics and underlying MDP are fully preserved when using the state-dependent action bounds procedure outlined above. Indeed, as the rescaling operation is a bijection, every original action is mapped to a unique normalized action (and vice versa). By constraining the policy to only support normalized actions within a fixed interval $[\tilde{a}_{\text{L}}\,;\tilde{a}_{\text{U}}]$, actions outside the range $[a_{\text{L}}(\boldsymbol{s})\,;a_{\text{U}}(\boldsymbol{s})]$ can however no longer be selected, leading to a pruned MDP from the agent's perspective.

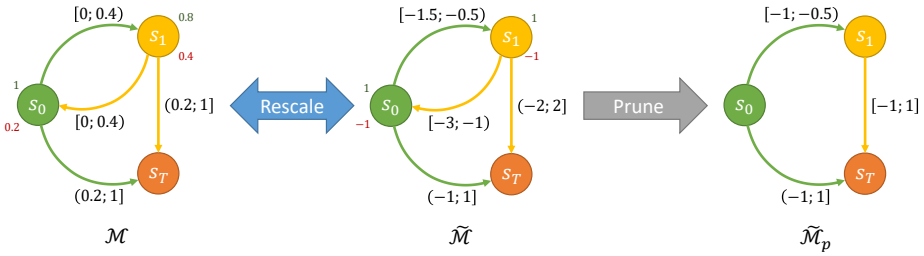

Fig. 4. Effect of the rescaling operation and imposed bounds on the actor network on the MDP of an example environment. The used rescaling function is σ_{lin} with bounds $a_{\text{L}}(\boldsymbol{s}_0) = 0.2, a_{\text{U}}(\boldsymbol{s}_0) = 1, a_{\text{L}}(\boldsymbol{s}_1) = 0.4, a_{\text{U}}(\boldsymbol{s}_1) = 0.8$.

To give some more intuition, let us consider the effect of state-dependent action bounds on a simple MDP as shown in Figure 4. The original MDP $\mathcal{M}$ consists of three states $\boldsymbol{s}_0$, $\boldsymbol{s}_1$ and $\boldsymbol{s}_T$. In every non-terminal state, the agent can select a continuous action a in the range $[0\,;1]$. Each arrow between two states denotes a transition potentially triggered by a certain range of actions. If an action can lead to multiple transitions, each such transition is equally likely.

Using the σ_{lin} rescaling function with bounds $a_{\text{L}}(\boldsymbol{s}_0) = 0.2$, $a_{\text{U}}(\boldsymbol{s}_0) = 1$, $a_{\text{L}}(\boldsymbol{s}_1) = 0.4$, $a_{\text{U}}(\boldsymbol{s}_1) = 0.8$, a rescaled MDP $\tilde{\mathcal{M}}$ can be constructed with normalized actions instead of the original actions. Because σ is a bijection, this normalized MDP is completely equivalent to the original one. By constraining the search for optimal policies to the set of policies with outputs in the interval $[-1\,;1]$, not every transition is still applicable. Hence, the normalized MDP can be further pruned to the final MDP $\tilde{\mathcal{M}}_p$ by removing all transitions corresponding to normalized actions outside the interval $[-1\,;1]$. This last pruning step is however not enforced on the environment itself but rather on the policies. In fact, all original transitions are still there, but some can no longer be selected by the constrained policies.

More formally we have the original MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \iota, \tau, r, \gamma)$, from which the rescaled MDP $\tilde{\mathcal{M}}$ is constructed — by introducing the rescaling function σ in the environment — as $(\mathcal{S}, \tilde{\mathcal{A}}, \iota, \tilde{\tau}, \tilde{r}, \gamma)$ with $\tilde{\mathcal{A}} = \{\tilde{\boldsymbol{a}} \mid \exists \boldsymbol{s} \in \mathcal{S}, \exists \boldsymbol{a} \in \mathcal{A} : \sigma(\tilde{\boldsymbol{a}}; \boldsymbol{s}) = \boldsymbol{a}\}$, $\tilde{\tau}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \tilde{\boldsymbol{a}}_t) : \mathcal{S} \times \mathcal{S} \times \tilde{\mathcal{A}} \to [0\,;1] = \tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \sigma(\tilde{\boldsymbol{a}}_t; \boldsymbol{s}_t))$ and $\tilde{r}(\boldsymbol{s}_t, \tilde{\boldsymbol{a}}_t, \boldsymbol{s}_{t+1}) : \mathcal{S} \times \tilde{\mathcal{A}} \times \mathcal{S} \to \mathbb{R} = r(\boldsymbol{s}_t, \sigma(\tilde{\boldsymbol{a}}_t; \boldsymbol{s}_t), \boldsymbol{s}_{t+1})$. The usage of bounded probability distributions and bounded activation functions in the actor network constrains the set of normalized policies as $\tilde{\Pi}_C = \{\tilde{\pi} \mid \forall \boldsymbol{s} \in \mathcal{S}, \forall \tilde{\boldsymbol{a}} \in \tilde{\mathcal{A}} : \tilde{\boldsymbol{a}}_{\text{L}} \leq \tilde{\boldsymbol{a}} \leq \tilde{\boldsymbol{a}}_{\text{U}} \vee \tilde{\pi}(\tilde{\boldsymbol{a}}|\boldsymbol{s}) = 0\}$. Hence, applying the chosen RL method to the experience tuples $(\boldsymbol{s}_t, \tilde{\boldsymbol{a}}_t, \tilde{r}_t, \boldsymbol{s}_{t+1})$ boils down to finding an optimal normalized policy as the solution of the constrained MDP

$$\tilde{\pi}^* = \arg\max_{\tilde{\pi} \in \tilde{\Pi}_C} \mathbb{E}_{\tilde{\pi}, \tilde{\tau}, \iota} \left[\sum_{k=0}^{\infty} \gamma^k \tilde{r}(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k, \boldsymbol{s}_{k+1}) \right]. \qquad (\tilde{\mathcal{M}} + \tilde{\Pi}_C)$$

This optimal *normalized* policy defines a corresponding optimal policy π^* in the original MDP $\mathcal{M}$ with added policy constraints $\Pi_C = \{\pi \mid \forall \boldsymbol{s} \in \mathcal{S}, \forall \boldsymbol{a} \in \mathcal{A} : \boldsymbol{a}_{\text{L}}(\boldsymbol{s}) \leq \boldsymbol{a} \leq \boldsymbol{a}_{\text{U}}(\boldsymbol{s}) \vee \pi(\boldsymbol{a}|\boldsymbol{s}) = 0\}$, i.e. the solution of the constrained MDP

$$\pi^* = \arg\max_{\pi \in \Pi_C} \mathbb{E}_{\pi, \tau, \iota} \left[\sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) \right]. \qquad (\mathcal{M} + \Pi_C)$$

As a result, we effectively found an optimal constrained policy, with variable support (based on the state-dependent action bounds). Figure 5 shows such transformed policies, after application of different rescaling functions. The transformed policies can be easily calculated from their normalized analogues using the 'change of variable' formula from probability theory (requiring the inverse rescaling function σ^{-1} to be differentiable) $\pi(a|\boldsymbol{s}) = \tilde{\pi}(\sigma^{-1}(a; \boldsymbol{s})|\boldsymbol{s}) \left| \partial_a \sigma^{-1}(a; \boldsymbol{s}) \right|$. Samples from this rescaled policy can be readily calculated using the procedure outlined in the previous two subsections, i.e. by transforming samples from the normalized policy using the chosen rescaling function σ.

As a final remark, let us briefly consider the case of a differentiable rescaling function, which allows to directly store the rescaled actions $\boldsymbol{a}$ in the replay buffer rather than the normalized actions. In such a situation, both the bounding and rescaling operations can be enforced on the policy (agent), leaving the environment untouched, effectively solving $(\mathcal{M} + \Pi_C)$ directly.

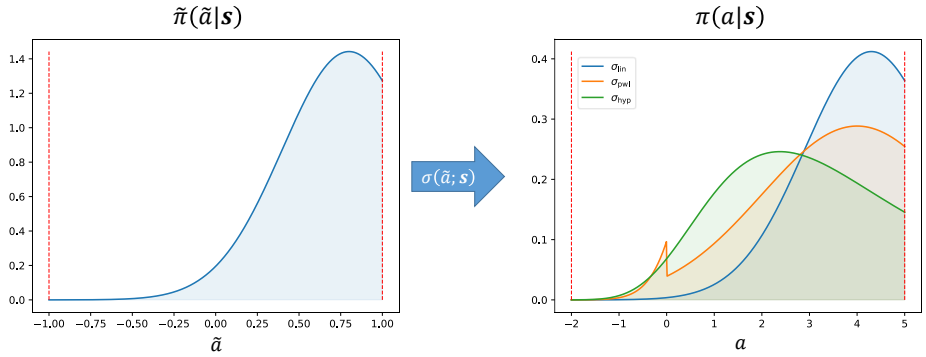

Fig. 5. Effect of the rescaling operation on the probability density function. On the left the normalized policy $\tilde{\pi}$ and on the right the rescaled policy π for different rescaling functions. The used bounds are $a_{\text{L}}(\boldsymbol{s}) = -2, a_{\text{U}}(\boldsymbol{s}) = 5$ with anchor point $a_0 = 0$ for $\tilde{a}_0 = 0$.

3.4 Implementation

A reference implementation of our approach in pytorch will be made available here[4]. In this implementation we specifically use the SAC method for learning, while the tanh function and truncated Gaussian distribution are used to constrain the policies. All of the discussed rescaling functions are available, allowing to easily reproduce the experiments of Appendix B.

4 Experimental Results

The effectiveness of the state-dependent action bounds is demonstrated in a virtual autonomous driving environment. A patent application for this specific application of state-dependent action bounds has been filed [10]. Additional experiments on two commonly used openAI gym environments can be found in Appendix B.

Every experiment is repeated ten times, using ten different seeds for initialization. Throughout training, E independent evaluation episodes are executed to get an estimate of the learned policy's average performance ($E = 5$ for the autonomous driving environment and $E = 10$ for the gym environments). These $10E$ datapoints are used to construct the plots (using exponential average smoothing with $\alpha = 0.1$) and report the performances.

4.1 Highway Driving

The considered simulated environment is a three lane highway in which a virtual driver (the agent) has to efficiently navigate traffic by issuing high-level steering

[4] https://github.com/dcbr/sdab

commands based on the current traffic situation in its neighbourhood. Through the continuous actions v_{ref} and d_{ref} the virtual driver can specify a desired longitudinal velocity and lateral reference position on the road, which are tracked by motion controllers. More details of the simulation environment can be found in Appendix C. The virtual driver's policy is trained using the TD3 method with added smoothness regularization [9], to increase passenger comfort.

Action Bounds In this environment, the state-dependent action bounds are used to avoid certain unsafe regions in state-action space.[5] Using prior knowledge of the environment, a safe maximum velocity and safe lateral position range can be determined, preventing collisions under reasonable worst case assumptions. Enforcing such state-dependent action bounds thus provides strong safety guarantees, allowing the agent to focus on its core task of efficiently navigating traffic.

Results Two constrained policies using the state-dependent action bounds with rescaling functions σ_{lin} and σ_{pwl} are trained.[6] Both constrained policies are compared with an unconstrained policy, which has to learn all aspects of driving on its own, using only the negative penalties in the reward signal obtained when the virtual driver chooses unreasonable or dangerous actions. Figure 6 summarizes the results of this comparison.

The plots show the average obtained reward and average episode length of each evaluation episode. By default, an episode lasts for 5000 timesteps, but the episode ends immediately when the virtual driver crashes. From these results, we can conclude both constrained policies are never involved in a crash throughout the whole training process, even at the very start. On the other hand, without guiding action bounds, the unconstrained policies cause many crashes early on and throughout the training process. Moreover, these crashes never truly vanish, even after extensive training. As a result, the best unconstrained policies never achieve the same level of performance as the constrained policies. Additionally, there is a significant difference in convergence speed. The constrained policies can focus on the 'core task' of efficiently navigating traffic, leading to a fast convergence after roughly $2 \cdot 10^5$ training steps. The unconstrained policies, on the other hand, have to simultaneously learn common traffic rules and how to avoid crashes, leading to a much slower convergence after roughly $8 \cdot 10^5$ training steps.

These results clearly highlight the benefits of state-dependent action bounds. First of all, it allows to effectively enforce *hard* constraints (bounds) on sampled policy actions, which are always satisfied, both during training and evaluation. Secondly, as the relevant domain knowledge can be enforced directly on the policies, it can keep reward functions simple, focusing on the core learning tasks. This not only considerably speeds up training, by only focusing on the relevant parts of state-action space, but can also result in better performing policies, as compared to unconstrained learning.

[5] Similar to the avoidance objective for the pendulum environment in Appendix B.
[6] These are the best performing rescaling functions in the experiments of Appendix B.

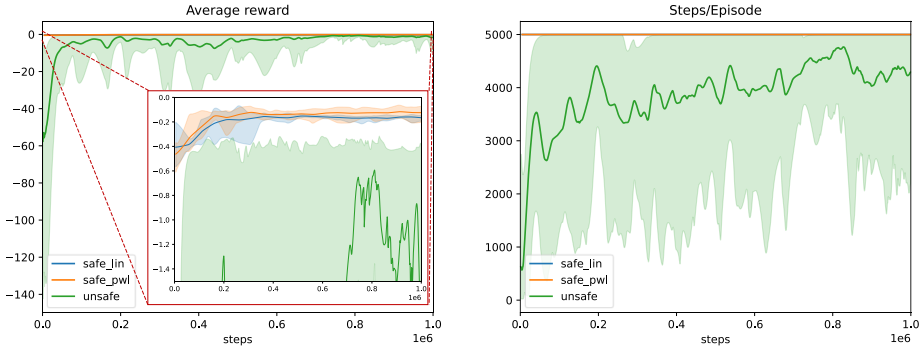

Fig. 6. Average obtained reward (left) and episode length (right) during evaluation as a function of training steps for two constrained policies with state-dependent safety bounds (blue and orange) and one unconstrained policy (green).

5 Conclusion

In this paper we proposed a novel way to enforce certain hard constraints on policies and prevent an agent from exploring and operating in irrelevant parts of the state-action space. More specifically, state-dependent action bounds allow the designer to restrict the range of available actions in certain states to a predefined (safe) interval. It is thus a tool to incorporate prior domain knowledge for a specific set of states, while keeping the model-free learning scheme's benefits. The proposed two-step procedure is both flexible and generic, making it easily combinable with existing policy-based and actor-critic reinforcement learning algorithms, as well as applicable to various domains. Experiments showed the effectiveness of the bounds, resulting in both faster learning and better performing policies.

A limitation of the current approach is the usage of a single interval of allowed actions for each state. In some situations there might however be multiple permissible action intervals. For example, an obstacle in front can be avoided by going either left or right. An extension to support multiple, disjoint action intervals might therefore be an interesting avenue for future research. Further extensions to the type of constraints that can be enforced is another potential path forward, for instance by allowing to bound the derivatives of policy actions.

Acknowledgements The presented results were primarily obtained under Ford Alliance Project KUL0076, funded by Ford. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. This research received funding from the Flemish Government (AI Research Program). Johan Suykens is affiliated to Leuven.AI - KU Leuven institute for AI, B-3000, Leuven, Belgium. The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

Bibliography

[1] Achiam, J., Held, D., Tamar, A., Abbeel, P.: Constrained policy optimization. In: 34th International Conference on Machine Learning, ICML 2017. vol. 1, pp. 30–47 (2017)

[2] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: 32nd AAAI Conference on Artificial Intelligence, AAAI 2018. pp. 2669–2678 (2018)

[3] Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe model-based reinforcement learning with stability guarantees. In: Advances in Neural Information Processing Systems. vol. 2017-Decem, pp. 909–919 (2017)

[4] Burkardt, J.: The Truncated Normal Distribution. Department of Scientific Computing , Florida State University pp. 1–35 (2014)

[5] Chandak, Y., Theocharous, G., Metevier, B., Thomas, P.S.: Reinforcement learning when all actions are not always available. In: AAAI 2020 - 34th AAAI Conference on Artificial Intelligence. pp. 3381–3388 (2020). https://doi.org/10.1609/aaai.v34i04.5740

[6] Cheng, R., Orosz, G., Murray, R.M., Burdick, J.W.: End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: 33rd AAAI Conference on Artificial Intelligence, AAAI 2019. pp. 3387–3395 (2019)

[7] Chow, Y., Nachum, O., Faust, A., Duenez-Guzman, E., Ghavamzadeh, M.: Lyapunov-based Safe Policy Optimization for Continuous Control (2019)

[8] Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., Tassa, Y.: Safe Exploration in Continuous Action Spaces (2018)

[9] De Cooman, B., Suykens, J., Ortseifen, A.: Improving temporal smoothness of deterministic reinforcement learning policies with continuous actions. In: 33rd Benelux Conference on Artificial Intelligence, BNAIC 2021. pp. 217–240 (Nov 2021)

[10] De Cooman, B., Suykens, J., Ortseifen, A., Subramanya, N.: Method for autonomous driving of a vehicle, a data processing circuit, a computer program, and a computer-readable medium, E.U. Patent Application EP22151063.9, filed 11 January 2022

[11] Figurnov, M., Mohamed, S., Mnih, A.: Implicit reparameterization gradients. In: Advances in Neural Information Processing Systems. vol. 2018-Decem, pp. 441–452 (2018)

[12] Fujimoto, S., Van Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. In: 35th International Conference on Machine Learning, ICML 2018. vol. 4, pp. 2587–2601 (2018)

[13] García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning (2015)

[14] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S.: Soft Actor-Critic Algorithms and Applications (2018)

[15] Hoel, C.J., Driggs-Campbell, K., Wolff, K., Laine, L., Kochenderfer, M.J.: Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving. Tech. rep. (2019)
[16] Kalweit, G., Huegle, M., Werling, M., Boedecker, J.: Deep Constrained Q-learning (2020)
[17] Kesting, A., Treiber, M., Helbing, D.: General lane-changing model MOBIL for car-following models. Transportation Research Record **1999**, 86–94 (2007). https://doi.org/10.3141/1999-10
[18] Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Advances in Neural Information Processing Systems. vol. 2015-Janua, pp. 2575–2583 (2015)
[19] Koller, T., Berkenkamp, F., Turchetta, M., Krause, A.: Learning-Based Model Predictive Control for Safe Exploration. In: Proceedings of the IEEE Conference on Decision and Control. vol. 2018-Decem, pp. 6059–6066 (jan 2019). https://doi.org/10.1109/CDC.2018.8619572
[20] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings (2016)
[21] Mirchevska, B., Pek, C., Werling, M., Althoff, M., Boedecker, J.: High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning. In: IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC. vol. 2018-Novem, pp. 2156–2162 (2018). https://doi.org/10.1109/ITSC.2018.8569448
[22] Rajamani, R.: Vehicle Dynamics and Control. Springer, 2nd edn. (2012). https://doi.org/10.1007/978-1-4614-1433-9
[23] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: 31st International Conference on Machine Learning, ICML 2014. vol. 1, pp. 605–619 (2014)
[24] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, A Bradford book, vol. 258. MIT Press, 1st edn. (1998)
[25] Tessler, C., Mankowitz, D.J., Mannor, S.: Reward constrained policy optimization. In: 7th International Conference on Learning Representations, ICLR 2019 (2019)
[26] Treiber, M., Hennecke, A., Helbing, D.: Congested traffic states in empirical observations and microscopic simulations. Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics **62**(2), 1805–1824 (2000). https://doi.org/10.1103/PhysRevE.62.1805
[27] Wachi, A., Sui, Y.: Safe reinforcement learning in constrained markov decision processes. In: 37th International Conference on Machine Learning, ICML 2020. vol. PartF16814, pp. 9739–9748 (2020)
[28] Zhang, H., Yu, T.: Taxonomy of reinforcement learning algorithms. In: Deep Reinforcement Learning: Fundamentals, Research and Applications, pp. 125–133. Springer, Singapore (jan 2020). https://doi.org/10.1007/978-981-15-4095-0_3

A Rescaling Functions

Let us first simplify notation, defining the rescaling functions as scalar bijections from x to y with parameters $\rho = \begin{bmatrix} x_{\mathrm{L}} & x_0 & x_{\mathrm{U}} & y_{\mathrm{L}} & y_0 & y_{\mathrm{U}} \end{bmatrix}^\top$,

$$\begin{aligned} \sigma(x;\rho) : [x_{\mathrm{L}}\,; x_{\mathrm{U}}] &\to [y_{\mathrm{L}}\,; y_{\mathrm{U}}], \\ y_{\mathrm{L}} &= \sigma(x_{\mathrm{L}};\rho), \\ y_{\mathrm{U}} &= \sigma(y_{\mathrm{U}};\rho), \end{aligned}$$

and the additional constraint $y_0 = \sigma(x_0;\rho)$ if the rescaling function supports an extra anchor point.

All of the rescaling functions introduced in Section 3 are translated and scaled *involutions*. An involution is a bijection whose inverse is equal to the function itself, i.e. $f(x) = f^{-1}(x)$ for all x in a certain set.

Lemma 1. *Denoting by $m(x;\tau_{\mathrm{y}},\sigma_{\mathrm{y}},\tau_{\mathrm{x}},\sigma_{\mathrm{x}}) = \tau_{\mathrm{y}} + \sigma_{\mathrm{y}} f(\sigma_{\mathrm{x}} x + \tau_{\mathrm{x}})$ the translated and scaled variant of an involution f, we can write its inverse as*

$$m^{-1}(y;\tau_{\mathrm{y}},\sigma_{\mathrm{y}},\tau_{\mathrm{x}},\sigma_{\mathrm{x}}) = m\left(y; \frac{-\tau_{\mathrm{x}}}{\sigma_{\mathrm{x}}}, \frac{1}{\sigma_{\mathrm{x}}}, \frac{-\tau_{\mathrm{y}}}{\sigma_{\mathrm{y}}}, \frac{1}{\sigma_{\mathrm{y}}}\right).$$

Proof. The inverse of m is given by

$$\begin{aligned} m^{-1}(y;\tau_{\mathrm{y}},\sigma_{\mathrm{y}},\tau_{\mathrm{x}},\sigma_{\mathrm{x}}) &= \frac{1}{\sigma_{\mathrm{x}}} f^{-1}\left(\frac{y-\tau_{\mathrm{y}}}{\sigma_{\mathrm{y}}}\right) - \frac{\tau_{\mathrm{x}}}{\sigma_{\mathrm{x}}} \\ &= \frac{-\tau_{\mathrm{x}}}{\sigma_{\mathrm{x}}} + \frac{1}{\sigma_{\mathrm{x}}} f\left(\frac{y}{\sigma_{\mathrm{y}}} + \frac{-\tau_{\mathrm{y}}}{\sigma_{\mathrm{y}}}\right) \\ &= m\left(y; \frac{-\tau_{\mathrm{x}}}{\sigma_{\mathrm{x}}}, \frac{1}{\sigma_{\mathrm{x}}}, \frac{-\tau_{\mathrm{y}}}{\sigma_{\mathrm{y}}}, \frac{1}{\sigma_{\mathrm{y}}}\right), \end{aligned}$$

where we used the property $f(x) = f^{-1}(x)$ of involutions.

Denoting by $P_{x\leftrightarrow y}$ the permutation matrix swapping x and y entries in ρ, i.e. $\tilde{\rho} = P_{x\leftrightarrow y}\rho = \begin{bmatrix} y_{\mathrm{L}} & y_0 & y_{\mathrm{U}} & x_{\mathrm{L}} & x_0 & x_{\mathrm{U}} \end{bmatrix}^\top$, we can prove the following Lemma.

Lemma 2. *Each of the considered rescaling functions σ_{lin}, σ_{pwl} and σ_{hyp} satisfy following relationship $\sigma^{-1}(y;\rho) = \sigma(y; P_{x\leftrightarrow y}\rho)$, allowing to easily calculate the inverse rescaling by swapping x and y entries in the parameter vector.*

Proof. The above relationship $\sigma^{-1}(y;\rho) = \sigma(y;\tilde{\rho})$ can be proven for each rescaling function by rewriting it as a scaled and translated involution

$$\sigma(x;\rho) = \tau_{\mathrm{y}}(\rho) + \sigma_{\mathrm{y}}(\rho) f(\sigma_{\mathrm{x}}(\rho) x + \tau_{\mathrm{x}}(\rho)),$$

satisfying

$$\tau_{\mathrm{y}}(\tilde{\rho}) = -\frac{\tau_{\mathrm{x}}(\rho)}{\sigma_{\mathrm{x}}(\rho)}, \quad \sigma_{\mathrm{y}}(\tilde{\rho}) = \frac{1}{\sigma_{\mathrm{x}}(\rho)}, \quad \tau_{\mathrm{x}}(\tilde{\rho}) = -\frac{\tau_{\mathrm{y}}(\rho)}{\sigma_{\mathrm{y}}(\rho)}, \quad \sigma_{\mathrm{x}}(\tilde{\rho}) = \frac{1}{\sigma_{\mathrm{y}}(\rho)}, \tag{3}$$

and applying Lemma 1.

Both linear rescaling functions σ_{lin}, σ_{pwl} can be rewritten as scaled and translated variants of the involution $f(x) = x$ using following parameters

$$\begin{aligned}
\sigma_{\text{lin}}: &\quad \tau_{\text{y}}(\rho) = y_{\text{L}}, \quad \sigma_{\text{y}}(\rho) = y_{\text{U}} - y_{\text{L}}, \quad \tau_{\text{x}}(\rho) = \frac{-x_{\text{L}}}{x_{\text{U}} - x_{\text{L}}}, \quad \sigma_{\text{x}}(\rho) = \frac{1}{x_{\text{U}} - x_{\text{L}}} \\
\sigma_{\text{pwl,L}}: &\quad \tau_{\text{y}}(\rho) = y_0, \quad \sigma_{\text{y}}(\rho) = y_{\text{L}} - y_0, \quad \tau_{\text{x}}(\rho) = \frac{-x_0}{x_{\text{L}} - x_0}, \quad \sigma_{\text{x}}(\rho) = \frac{1}{x_{\text{L}} - x_0} \\
\sigma_{\text{pwl,U}}: &\quad \tau_{\text{y}}(\rho) = y_0, \quad \sigma_{\text{y}}(\rho) = y_{\text{U}} - y_0, \quad \tau_{\text{x}}(\rho) = \frac{-x_0}{x_{\text{U}} - x_0}, \quad \sigma_{\text{x}}(\rho) = \frac{1}{x_{\text{U}} - x_0}
\end{aligned}$$

for which the conditions (3) can be readily checked. Note that the inverse of the piecewise function σ_{pwl} is given by the inverse of its constituents in each of their non-overlapping domains (both σ_{pwl} and its constituents are bijections).

Similarly, the hyperbolic rescaling function σ_{hyp} can be rewritten as a transformed version of the involution $f(x) = \frac{1}{x}$ using following scales and translations

$$\begin{aligned}
\tau_{\text{y}}(\rho) &= \frac{(y_{\text{U}} - y_0)(x_{\text{L}} - x_0)y_{\text{L}} - (y_{\text{L}} - y_0)(x_{\text{U}} - x_0)y_{\text{U}}}{(y_{\text{U}} - y_0)(x_{\text{L}} - x_0) - (y_{\text{L}} - y_0)(x_{\text{U}} - x_0)}, \\
\sigma_{\text{y}}(\rho) &= \frac{(y_{\text{U}} - y_0)(y_{\text{L}} - y_0)(y_{\text{U}} - y_{\text{L}})}{(y_{\text{U}} - y_0)(x_{\text{L}} - x_0) - (y_{\text{L}} - y_0)(x_{\text{U}} - x_0)}, \\
\tau_{\text{x}}(\rho) &= \frac{(y_{\text{U}} - y_0)(x_{\text{L}} - x_0)x_{\text{U}} - (y_{\text{L}} - y_0)(x_{\text{U}} - x_0)x_{\text{L}}}{(x_{\text{U}} - x_0)(x_{\text{L}} - x_0)(x_{\text{U}} - x_{\text{L}})}, \\
\sigma_{\text{x}}(\rho) &= -\frac{(y_{\text{U}} - y_0)(x_{\text{L}} - x_0) - (y_{\text{L}} - y_0)(x_{\text{U}} - x_0)}{(x_{\text{U}} - x_0)(x_{\text{L}} - x_0)(x_{\text{U}} - x_{\text{L}})},
\end{aligned}$$

for which the conditions (3) can be verified.

B State-Dependent Bounds in Gym Environments

To provide some intuition, we compare different implementations of the state-dependent action bounds on two commonly used openAI gym environments[7]: `Pendulum-v0` and `LunarLanderContinous-v2`. For these experiments, we used a customized version of the latest Stable-Baselines implementation[8].

B.1 Inverted Pendulum

The objective in this `Pendulum-v0` environment is to swing up and stabilize an inverted pendulum around the upward position. The observable state by the agent $\boldsymbol{s} = \left[\cos(\theta)\ \sin(\theta)\ \dot{\theta}\right]^\top$ contains information about the pendulum's orientation angle θ and angular velocity $\dot{\theta}$. The scalar action a denotes the torque to apply on the pendulum and is bounded by the interval $[-2\,;2]$ (larger or smaller values are clipped by the environment). We consider two different applications of state-dependent action bounds on this environment.

[7] https://gym.openai.com
[8] https://github.com/DLR-RM/stable-baselines3

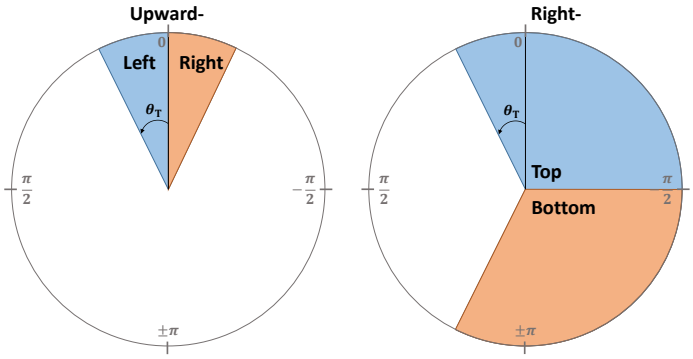

Fig. 7. Visualization of the state space (orientation angle component) partitioning imposed by the Upward and Right predicates in the `Pendulum-v0` environment.

Aiding Stabilization The first set of state-dependent action bounds aims to aid in the stabilization of the pendulum around the upward position. As prior domain knowledge, we know that at the upward equilibrium point, the velocity of the pendulum should be low and small deviations to the left or right should be corrected by a negative or positive torque respectively. This extra domain knowledge is encoded in the chosen state-dependent bounds of this setup. Using the resulting bounds, the agent still has to figure out how to swing up the pendulum, but once the pendulum is close to the upward position, tighter action bounds aid in the stabilization by slowing down the pendulum and preventing it from falling over.

$$a_{\mathrm{L}}(\boldsymbol{s}) = \begin{cases} 2-\epsilon & \text{if } \mathrm{Upward}(\boldsymbol{s}) \wedge \mathrm{FastNeg}(\boldsymbol{s}) \quad \textit{(S.1)} \\ -(2-\epsilon)\frac{\sin(\theta)}{\theta_{\mathrm{T}}} & \text{if } \mathrm{UpwardRight}(\boldsymbol{s}) \wedge \mathrm{Slow}(\boldsymbol{s}) \quad \textit{(S.2)} \\ -2 & \text{else} \end{cases}$$

$$a_{\mathrm{U}}(\boldsymbol{s}) = \begin{cases} -2+\epsilon & \text{if } \mathrm{Upward}(\boldsymbol{s}) \wedge \mathrm{FastPos}(\boldsymbol{s}) \quad \textit{(S.1)} \\ (-2+\epsilon)\frac{\sin(\theta)}{\theta_{\mathrm{T}}} & \text{if } \mathrm{UpwardLeft}(\boldsymbol{s}) \wedge \mathrm{Slow}(\boldsymbol{s}) \quad \textit{(S.2)} \\ 2 & \text{else} \end{cases}$$

with predicates

$$\begin{aligned} \mathrm{UpwardLeft}(\boldsymbol{s}) &= \cos(\theta) > 0 \wedge 0 \leq \sin(\theta) < \theta_{\mathrm{T}}, \\ \mathrm{UpwardRight}(\boldsymbol{s}) &= \cos(\theta) > 0 \wedge -\theta_{\mathrm{T}} < \sin(\theta) \leq 0, \\ \mathrm{Upward}(\boldsymbol{s}) &= \mathrm{UpwardLeft}(\boldsymbol{s}) \vee \mathrm{UpwardRight}(\boldsymbol{s}), \\ \mathrm{FastNeg}(\boldsymbol{s}) &= \dot{\theta} < -\dot{\theta}_{\mathrm{T}}, \\ \mathrm{FastPos}(\boldsymbol{s}) &= \dot{\theta} > \dot{\theta}_{\mathrm{T}}, \\ \mathrm{Slow}(\boldsymbol{s}) &= \neg\mathrm{FastNeg}(\boldsymbol{s}) \wedge \neg\mathrm{FastPos}(\boldsymbol{s}). \end{aligned}$$

The *slowdown* conditions and bounds *(S.1)* ensure the pendulum is slowed down when it is approaching the upward position, while the *stabilization* conditions and bounds *(S.2)* attempt to prevent the pendulum from falling over when it is close to the upward equilibrium position. See the left side of Figure 7 for a visualization of the state space partitioning originating from the Upward predicates. In the conducted experiments we chose $\epsilon = 0.1$, $\theta_{\mathrm{T}} = 0.3$ and $\dot{\theta}_{\mathrm{T}} = 0.3$.

Avoiding One Side For this environment we also analyze a second set of state-dependent action bounds to show their usage in avoiding certain (e.g. unsafe) regions in state space. More specifically, in this setup the chosen bounds push the pendulum away from the right side[9], leaving only the left side available for swinging up the pendulum.

$$a_{\mathrm{L}}(\boldsymbol{s}) = \max \begin{cases} (2-\epsilon)\left(2\frac{\min\{-\dot{\theta},\dot{\theta}_{\mathrm{M}}\}}{\dot{\theta}_{\mathrm{M}}} - 1\right) & \text{if RightTop}(\boldsymbol{s}) \wedge \text{FastNeg}(\boldsymbol{s}) \quad \textit{(A.1)} \\ (2-\epsilon)\frac{\max\{\sin(\theta),-\theta_{\mathrm{T}}\}}{\theta_{\mathrm{T}}} & \text{if RightTop}(\boldsymbol{s}) \quad \textit{(A.2)} \\ -2 & \text{else} \end{cases}$$

$$a_{\mathrm{U}}(\boldsymbol{s}) = \min \begin{cases} (-2+\epsilon)\left(2\frac{\min\{\dot{\theta},\dot{\theta}_{\mathrm{M}}\}}{\dot{\theta}_{\mathrm{M}}} - 1\right) & \text{if RightBottom}(\boldsymbol{s}) \wedge \text{FastPos}(\boldsymbol{s}) \quad \textit{(A.1)} \\ (-2+\epsilon)\frac{\max\{\sin(\theta),-\theta_{\mathrm{T}}\}}{\theta_{\mathrm{T}}} & \text{if RightBottom}(\boldsymbol{s}) \quad \textit{(A.2)} \\ 2 & \text{else} \end{cases}$$

with extra predicates

$$\begin{aligned} \text{RightTop}(\boldsymbol{s}) &= \sin(\theta) < \theta_{\mathrm{T}} \wedge \cos(\theta) > 0, \\ \text{RightBottom}(\boldsymbol{s}) &= \sin(\theta) < \theta_{\mathrm{T}} \wedge \cos(\theta) \leq 0, \\ \text{Right}(\boldsymbol{s}) &= \text{RightTop}(\boldsymbol{s}) \vee \text{RightBottom}(\boldsymbol{s}), \end{aligned}$$

where we used the special notation min{ and max{ to denote taking the minimum/maximum of any (possibly multiple) active cases. The *slowdown* conditions and bounds *(A.1)* ensure the pendulum is slowed down when it is approaching the right side with high velocity, while the *avoidance* conditions and bounds *(A.2)* force the pendulum away from the right side. See the right side of Figure 7 for a visualization of the state space partitioning caused by the Right predicates. In the conducted experiments we chose $\epsilon = 0.1$, $\theta_{\mathrm{T}} = 0.3$, $\dot{\theta}_{\mathrm{T}} = 0$ and $\dot{\theta}_{\mathrm{M}} = 6$.

Results Figure 8 shows the average evaluation return under both setups. Different SAC policies with state-dependent action bounds (using different rescaling functions) are compared against an unconstrained SAC policy. In this relatively simple environment there are no significant differences between the different rescaling functions, as all policies solve the environment in less than 50000 training steps.

[9] Leaving only some slack for the swing up movements, as otherwise, with too strict bounds, the environment is unsolvable.

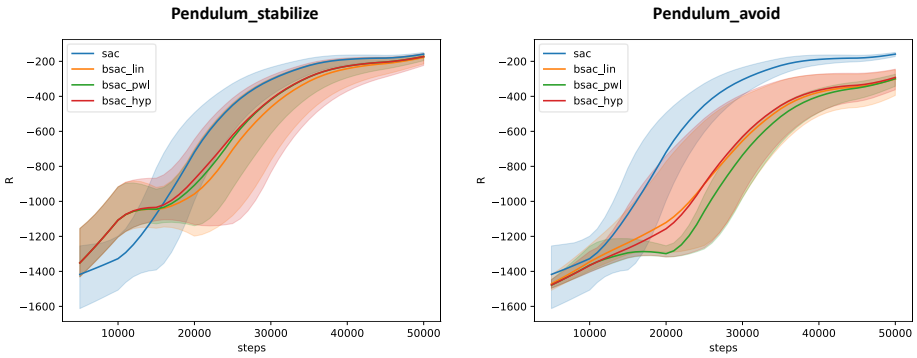

Fig. 8. Average evaluation return on the `Pendulum-v0` environment using the stabilization (left) and avoidance (right) state-dependent bounds for various rescaling functions.

While the stabilizing bounds provide an initial benefit to the policies (giving higher rewards in the initial training phase), this does not lead to faster training convergence (left plot). The constrained policies require extra training time to figure out the effect of the state-dependent action bounds on the chosen normalized actions. As the unconstrained SAC policy converges already very fast in this environment, there is little benefit in aiding the stabilization process through state-dependent action bounds.

On the right plot of Figure 8 we can also notice the slightly lower performance of the constrained policies after convergence using the avoidance setup. Such a drop is to be expected as we are effectively solving a harder problem. For certain initial states, it would be faster to swing up along the right side, but this is made impossible by the action bounds. Hence, the constrained policies take a longer time to reach the upward position for certain initial states, which results in a slightly lower return. The most important question under this setup is however if the bounds actually succeeded in avoiding the right side as much as possible. Figure 9 shows that this is indeed the case. The unconstrained policies roughly utilize the left and right side equally, as the left state histogram is almost symmetric about the vertical radial axis. On the other hand, the right state histogram shows that the constrained policies only use the right side for swing up (high velocities in the lower right quadrant) and to move away from initial states[10] (low velocities in the upper right quadrant).

B.2 Lunar Lander

The agent's objective in the `LunarLanderContinuous-v2` environment is to safely land a lunar module within a marked landing zone on the moon's surface in a simplified 2D setting. The observable state of the agent $\boldsymbol{s}$ consists, among other signals, of the lander's downward velocity v, its orientation angle θ and

[10] Initial states are not enforced to lie on the left side.

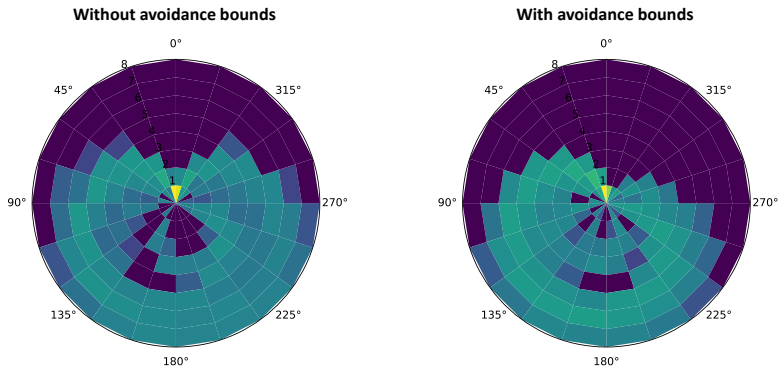

Fig. 9. Polar histogram of the pendulum's state space using states visited by all evaluation episodes at the end of training. The angular axis represents the pendulum's orientation angle θ, while the radial axis represents the pendulum's angular velocity $\dot{\theta}$. The left plot shows the results for an unconstrained SAC policy, while the right plot shows the results for a constrained SAC policy, using σ_{pwl} as the rescaling function.

angular velocity $\dot{\theta}$. The action vector $\boldsymbol{a}$ consists of two components bounded to the interval $[-1\,;1]$ (imposed by the environment). The first component handles the main engine (at the module bottom), while the second component controls the two side engines (at the left and right). The relative power $P \in [0\,;1]$ of each engine is determined as follows

$$P_{\text{main}} = \begin{cases} 0 & -1 \leq a_1 \leq 0 \\ \frac{1+a_1}{2} & 0 < a_1 \leq 1 \end{cases},$$

$$P_{\text{left}} = \begin{cases} 0 & -1 \leq a_2 \leq 0.5 \\ a_2 & 0.5 < a_2 \leq 1 \end{cases}, \qquad P_{\text{right}} = \begin{cases} -a_2 & -1 \leq a_2 < -0.5 \\ 0 & -0.5 \leq a_2 \leq 1 \end{cases}.$$

Stabilizing Bounds The left plot in Figure 10 shows the fraction of evaluation episodes ending either in a crash or astray (too far from the landing zone) for an unconstrained SAC policy. As can be seen, at the start of training, most evaluation episodes end without a correct landing on the lunar module's legs. To help the agent in correctly landing on its legs, stabilizing state-dependent bounds are enforced on its actions. Similar to the pendulum environment these bounds ensure the lunar module does not tilt too much to the left or right by activating the appropriate side engines. Additionally, the bounds ensure the main engine is activated when the downward velocity exceeds a certain threshold value.

The bounds on the first action component ensure the main engine is turned on whenever the downward velocity is too large, thereby slowing down the descent.

$$a_{\text{L},1}(\boldsymbol{s}) = \begin{cases} \epsilon & \text{if } v > v_{\text{T}} \\ -1 & \text{else} \end{cases} \qquad\qquad a_{\text{U},1}(\boldsymbol{s}) = 1$$

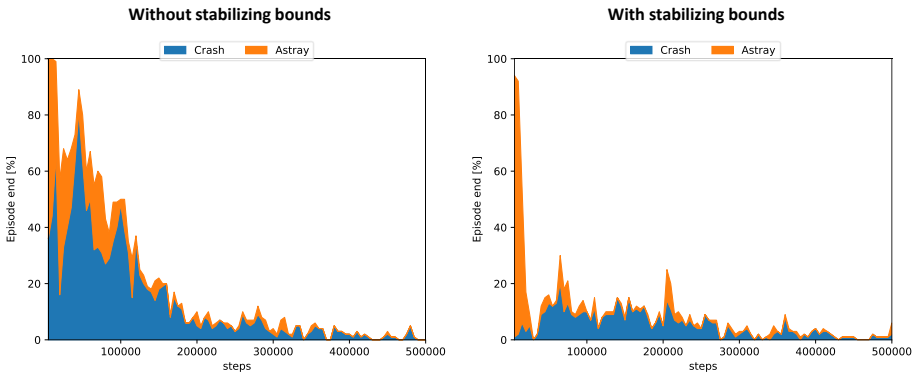

Fig. 10. Fraction of evaluation episodes ending in a crash (excessive downward velocity on touchdown or not landed on legs) or astray (too far away from the landing zone) for an unconstrained SAC policy (left) and a SAC policy with stabilizing action bounds, using σ_{lin} as the rescaling function (right).

The bounds on the second action component ensure the side engines are turned on when the lander tilts too much to the left or right without adjusting for it, thereby stabilizing the landing.

$$a_{\text{L},2}(\boldsymbol{s}) = \begin{cases} 0.5 + \epsilon & \text{if TiltLeft}(\boldsymbol{s}) \wedge \neg\text{SpinRight}(\boldsymbol{s}) \\ -1 & \text{else} \end{cases}$$

$$a_{\text{U},2}(\boldsymbol{s}) = \begin{cases} -0.5 - \epsilon & \text{if TiltRight}(\boldsymbol{s}) \wedge \neg\text{SpinLeft}(\boldsymbol{s}) \\ -1 & \text{else} \end{cases}$$

using predicates

$$\text{TiltLeft}(\boldsymbol{s}) = \theta > \theta_{\text{T}}, \qquad \text{TiltRight}(\boldsymbol{s}) = \theta < -\theta_{\text{T}},$$
$$\text{SpinLeft}(\boldsymbol{s}) = \dot{\theta} > \dot{\theta}_{\text{T}}, \qquad \text{SpinRight}(\boldsymbol{s}) = \dot{\theta} < -\dot{\theta}_{\text{T}}.$$

In the conducted experiments we used $\epsilon = 0.01$, $v_{\text{T}} = 0.2$, $\theta_{\text{T}} = 0.2$ and $\dot{\theta}_{\text{T}} = 0.12$. With these bounds in place, the resulting constrained policies have significantly fewer evaluation episodes ending in a crash or astray (right plot in Figure 10). The remaining job for the agent is now to steer the lunar module towards the landing zone and prevent any remaining crashes[11].

Results Figure 11 compares the average evaluation returns of different constrained SAC policies with an unconstrained SAC policy. Similar to the results on the pendulum environment, the constrained policies have a head start. The

[11] The chosen bounds do not guarantee a crash-free experience, but could be further tightened if needed.

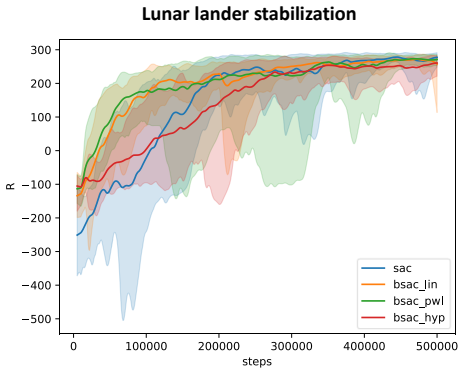

Fig. 11. Average evaluation return on the `LunarLanderContinuous-v2` environment using the stabilizing state-dependent bounds for various rescaling functions.

imposed state-dependent bounds prevent most of the crashes, giving higher returns early on in the training process. Contrary to the pendulum results, the extra stabilization benefit of most constrained policies *does* lead to a faster overall convergence in this case. The state-dependent bounds allow the agent to focus on the most relevant parts of the state-action space during training and exploration, instead of loosing many timesteps on the irrelevant parts that lead to crashes. Only the hyperbolic rescaling function σ_{hyp} cannot turn its initial head start in a faster overall convergence.

B.3 SAC Hyperparameters

The tables below show the used hyperparameters for the SAC algorithm on the two used OpenAI gym environments. Most of these values correspond to the tuned hyperparameters of the Stable-Baselines3 repository[12], the differences are highlighted in bold.

Table 1. Overview of the used hyperparameters for each environment. The shown hyperparameters are: maximum timesteps per episode k_M, total training timesteps $k_M \cdot T_M$, discount factor γ, replay buffer size $|\mathcal{B}|$, learning rate (for both actor and critic networks) η.

Environment	$\boldsymbol{k_M}$	$\boldsymbol{k_M T_M}$	$\boldsymbol{\gamma}$	$\boldsymbol{\lvert\mathcal{B}\rvert}$	$\boldsymbol{\eta}$
LunarLanderContinuous-v2	1000	$5 \cdot 10^5$	0.99	$1 \cdot 10^6$	$\mathbf{7 \cdot 10^{-4}}$
Pendulum-v0	200	$\mathbf{5 \cdot 10^4}$	**0.98**	$\mathbf{2 \cdot 10^5}$	$1 \cdot 10^{-3}$

[12] https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/hyperparams/sac.yml

Table 2. Overview of the used hyperparameters, common across both environments.

Common hyperparameters		
Warmup timesteps		10000
Batch size	B	256
Polyak averaging constant	τ	$5 \cdot 10^{-3}$
Network architecture – hidden dimensions (actor + critic)		400×300
Entropy coefficient	α	Automatic adjustment

C Autonomous Highway Driving Environment

The results shown in Section 4 for the highway driving environment are obtained using a proprietary highway simulator. In this section the most relevant components of this simulator are briefly discussed.

C.1 Roads

All experiments were conducted on a three lane highway, shaped as a closed-loop circuit, with both straight and curved segments. The maximum speed limit was set to 30m/s in all lanes, although some vehicles were instructed to slightly deviate from this limit, to get more varying situations on the road.

C.2 Vehicles

Every vehicle in the simulator follows the kinematic bicycle model (KBM) [22] to update its state based on the selected inputs

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ \frac{v}{l_r}\sin\beta \\ a \\ \cos\beta \end{bmatrix} \quad \beta = \arctan\left(\frac{l_r}{l_f+l_r}\tan\delta\right).$$

The vehicle's local state vector consists of an absolute x and y position, a heading angle ψ and velocity v. The vehicle can be controlled through its inputs, consisting of a steering angle δ and a longitudinal acceleration a. To make the control task of the virtual driver (agent) easier, extra vehicle motion controllers are used to stabilize the vehicle on the road, allowing the agent to select high-level steering actions $\boldsymbol{a}$, consisting of a desired longitudinal velocity v_{ref} and desired lateral position d_{ref}, to solve the driving task. To take correct high level steering decisions, the virtual driver needs some extra information about other traffic participants in its neighbourhood. This information is gathered in the agent's observation vector $\boldsymbol{s}$, containing local information such as the vehicle's offset w.r.t. different lane centers and its velocity components; and relative information such as relative distances and velocities w.r.t. neighbouring traffic.

C.3 Policies

Every vehicle is controlled by a policy, mapping observations $\boldsymbol{s}$ to suitable high-level actions $\boldsymbol{a}$. The policy of the autonomous vehicle is learned using any of the described reinforcement learning methods in this paper. The policies of the other vehicles in the simulation environment are fixed beforehand. A mixture of vehicles equipped with a custom rule-based policy and a policy implementing the 'Intelligent Driver Model' (IDM) [26] and 'Minimizing Overall Braking Induced by Lane change' (MOBIL) [17] is used. Both policies try to mimic rudimentary human driving behaviour, although being fully deterministic.

C.4 Reward

The used reward signal is calculated as a weighted sum of different penalties

$$r = \frac{w_{\mathrm{F}} r_{\mathrm{F}} + w_{\mathrm{V}} r_{\mathrm{V}} + w_{\mathrm{C}} r_{\mathrm{C}} + w_{\mathrm{R}} r_{\mathrm{R}}}{w_{\mathrm{F}} + w_{\mathrm{V}} + w_{\mathrm{C}} + w_{\mathrm{R}}}.$$

The first 'frontal' component r_{F} gives a penalty whenever the following distance to the leading vehicle is smaller than a predefined threshold. The 'velocity' component r_{V} gives a penalty whenever the virtual driver is not travelling at or near the maximum allowed speed. The third 'center' component r_{C} gives a penalty whenever the vehicle is not travelling central in the lane. To force the virtual driver to keep right whenever possible, the 'right' penalty r_{R} is given whenever there is a free lane to the right available. Finally, an extra penalty is given when the virtual driver collides with other vehicles or the highway boundaries.

C.5 Action Bounds

In this environment, the state-dependent action bounds are used to avoid potentially unsafe regions in state-action space. To determine the bounds, following braking criterion is used

$$\min\left(\Delta x, \Delta x + \frac{v_{\mathrm{L}}^2}{2b} - \frac{v_{\mathrm{F}}^2}{2b}\right) > \Delta x_{\mathrm{SAFE}}, \tag{4}$$

where Δx is the following distance between 2 vehicles, v_{L} and v_{F} are the velocities of the leading and following vehicle respectively, b is the maximum deceleration of both vehicles and Δx_{SAFE} is a minimum safe distance to keep between both vehicles. For simplicity we assume following vehicles can react instantly (no delay) to changing behaviour of the leading vehicle and the maximum deceleration b is assumed to be the same for both leading and following vehicles. Complying with the braking criterion then ensures the following vehicle is always able to avoid a crash with the leading vehicle, even in case of an emergency brake (braking with maximum deceleration b until standstill).

The upper bound for the longitudinal reference velocity can be derived from (4) as

$$v_{\mathrm{U}} = \sqrt{v_{\mathrm{L}}^2 + 2b(\Delta x - \Delta x_{\mathrm{SAFE}})}.$$

This expression is evaluated for all visible[13] leading vehicles having a lateral overlap with the virtual driver (Figure 12.a). The lowest encountered value for v_{U} is then taken as the final velocity upper bound.

For the bounds on the lateral reference position, let us first consider the set of safe lateral intervals $\mathcal{D}_{\mathrm{SAFE}} = \{[d_{\mathrm{l}}\,;d_{\mathrm{r}}] \subset [d_{\mathrm{left}}\,;d_{\mathrm{right}}] \mid \forall d \in [d_{\mathrm{l}}\,;d_{\mathrm{r}}] : (4) \text{ holds}\}$ where d_{left} and d_{right} denote the lateral distance to the left and right road boundary respectively. The braking criterion is evaluated for all visible[13] vehicles having some overlap with lateral position d, with the virtual driver as leading or following vehicle depending on the other vehicle's longitudinal position with respect to the virtual driver. The bounds for the lateral reference position, are then constructed such that $[d_{\mathrm{L}}\,;d_{\mathrm{U}}] \in \mathcal{D}_{\mathrm{SAFE}}$ is the nearest (with respect to the virtual driver's current lateral position) and largest safe interval (Figure 12.b).

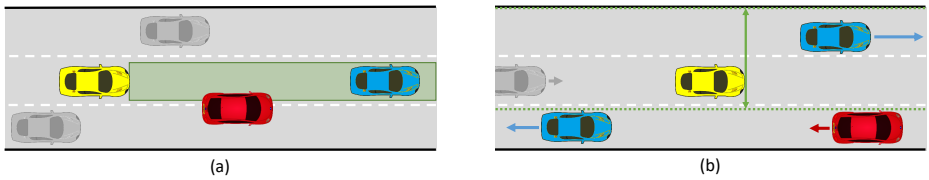

Fig. 12. Illustration of safe action bound determination on highways using the braking criterion (4). The yellow car is driven by the virtual driver; the gray cars are not considered for determining the safe action range; and the red cars are the limiting entities, responsible for the lowest velocity upper bound or tightest lateral action range. The arrows indicate the relative velocity of each car with respect to the virtual driver's velocity. The left side (a) illustrates the determination of the longitudinal velocity upper bound, by evaluating the braking criterion for all leading vehicles with lateral overlap (green area). The right side (b) illustrates the determination of the safe lateral interval, by evaluating the braking criterion for each visible car. The green lines indicate the resulting safe action range: allowing the virtual driver to move towards the left lane — as the only vehicle there is quickly moving away from the virtual driver — but preventing movement towards the right lane — as this lane is occupied by the slower red vehicle.

[13] Visible means part of the state vector in this context.