Improving temporal smoothness of deterministic reinforcement learning policies with continuous actions

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. A commonly observed weakness of deterministic reinforcement learning policies with continuous action spaces, such as those obtained after training with the DDPG or TD3 methods, is the temporal roughness of their output signals (chosen actions). This is a serious deterrent for real-life application of such policies in continuous control tasks. For instance, in autonomous driving the rate of change of lateral acceleration is typically restricted to ensure passenger safety and comfort. Therefore, we propose a set of modified TD3 algorithms to improve the temporal smoothness of the trained agent's chosen actions. These smoothed TD3 (STD3) algorithms can be applied to smoothen policies; either in a post-processing training phase, or from the very start of training in an attempt to reduce the roughness cost (constraint) to an acceptable level throughout training. The proposed methodology is applied to some well-known benchmark environments, as well as to a more complex autonomous driving problem. Results show a consistent reduction of roughness without significant performance deterioration.

Keywords: Smooth Control · Reinforcement Learning · Deterministic Policies · Autonomous Driving.

1 Introduction

The usage of deterministic policies with continuous action spaces can lead to very oscillatory system behavior (see Figure 2). Such behavior is typically characterized by control signals with large, altering gradients in the time domain and high frequency components in the frequency domain. Although not as much of a problem in virtual simulations, this can severely impact the applicability of the learned policies in the real world, where such jerky control signals might wear down or damage critical components.

Mitigation. For simple or purely virtual environments such roughness issues could be dealt with using mitigation strategies. One option is to redefine the

action space and switch to derivative control of the system [4, 23, 17]. This way, the rough derivative actions will be smoothed out by the extra integrators in the environment's dynamics. Alternatively, the rough action signals could be low-pass filtered, effectively damping high frequency oscillations [8]. Another commonly used mitigation strategy in reinforcement learning consists of adding a roughness penalty in the reward signal [4, 23, 15], making it beneficial for the agent to select actions that do not change too much from one timestep to the next. While such techniques may work on relatively simple environments, they quickly become cumbersome in more realistic setups. In this paper, we try to tackle the roughness problem at its core, by embedding the smoothness constraints into the training process, leading to a set of smoothed TD3 (STD3) algorithms. These algorithms lead to smoother policies and simpler models (no unnecessary integrators and filters, less convoluted reward signals), allowing the designer to focus more on prime objectives instead.

Smooth exploration. Smoothness issues with neural network outputs have been addressed before [6] to increase the network's generalization capabilities. A recent overview of existing smoothing techniques for neural networks and their advantages is given by Rosca et al. [16]. In optimal control, the requirement of smooth control signals has been dealt with, e.g. under the form of slew rate constraints. It is thus surprising that such techniques have been rarely applied to the reinforcement learning domain. Initial attempts mostly focused on the smoothness of the exploratory policy during training. Lillicrap et al. [11] suggested the usage of autocorrelated Ornstein-Uhlenbeck noise to guarantee proper exploration of the state-action space when working with deterministic policies. Under small time discretizations, the rough uncorrelated Gaussian noise samples could cancel each other out, leading to insufficient exploration and suboptimal learned policies. Raffin et al. [14] presented generalized state-dependent exploration (gSDE) as another solution for the non-smoothness of Gaussian noise samples. By making the noise function state-dependent through a linear combination of policy features and fixing the linear weights for a given amount of training steps, the smoothness of the behavioural policy is drastically improved. The prime focus of such techniques lies on the smoothness of policies during training and exploration, while our focus in this paper lies on the smoothness of the learned policies during evaluation or deployment. Hence these methods could be seen as an orthogonal approach and could be readily combined with our proposed smoothed TD3 variants to improve the *overall smoothness*, during both training and evaluation.

Regularization. The usage of output regularization, in combination with derivative control, has been investigated by Chisari et al. [4]. By forcing the action rates to remain small, the integrated actions that are passed to the environment's dynamics remain smooth. Recently, Mysore et al. [12] introduced 'Conditioning for Action Policy Smoothness' (CAPS), a method to improve temporal and spatial smoothness of policies through the addition of two regularization terms on

the policy network. This smoothness regularization is also leveraged by our proposed STD3 methods, albeit in a more generic setting. In fact, the temporal smoothness regularization of CAPS corresponds to the specific $\text{STD3}_{\text{C,fix}}$ variant, introduced here. Spatial smoothness is not further considered in this work, but could also be accounted for using an extra regularization term and a dedicated spatial smoothing schedule.

This paper is organized as follows. In Section 2 the required reinforcement learning (RL) preliminaries are described, followed by a short motivational example in Section 3 to highlight the importance of additional smoothness constraints. Section 4 proceeds by introducing the different smoothed TD3 variants, used to improve the learned policy's temporal smoothness. Finally, the different variants are evaluated and compared on different environments in Section 5. These experiments show the great potential of the added smoothness constraints, as they not only drastically improve the policy's smoothness, but also outperform standard TD3 policies on a majority of the investigated environments.

2 Reinforcement Learning

In model-free Reinforcement Learning (RL) the objective is to find an optimal controller (policy) for an entity (agent) acting under a-priori unknown system dynamics (an unknown environment). At any given point in time t, the agent has access to the current state $\boldsymbol{s}_t \in \mathcal{S}$ of the environment; or an observation of this state if the system is only partially observable. The controller then maps these states $\boldsymbol{s}_t$ to suitable actions $\boldsymbol{a}_t \in \mathcal{A}$ and is often referred to as the agent's *policy*. The execution of an action, will bring the agent to a new state $\boldsymbol{s}_{t+1}$ — following the system dynamics — after which the same procedure can be repeated. To improve its policy, the agent has one extra source of information available: the reward signal $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ which describes how favourable it was to select action $\boldsymbol{a}_t$ while being in state $\boldsymbol{s}_t$ and transitioning to state $\boldsymbol{s}_{t+1}$. The optimal controller is thus the one which maximizes the agent's long term accumulated reward.

More formally, the RL problem can be described as the Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \sigma_0, \tau, r, \gamma)$ with state space $\mathcal{S} \subset \mathbb{R}^S$, action space $\mathcal{A} \subset \mathbb{R}^A$, initial-state distribution $\sigma_0(\boldsymbol{s}_0) : \mathcal{S} \to [0\,;1]$, state-transition distribution $\tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0\,;1]$, reward signal $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and discount factor $\gamma \in [0\,;1)$. Note that the stochastic environment dynamics τ, σ_0 are modelled as a probability distribution but remain unknown to the agent. Usually, the agent's policy is also modelled through a probability distribution $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ from which suitable actions can be sampled at every timestep. The special case of deterministic policies can also be considered $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$. Learning the optimal policy π^* under such a framework then corresponds to finding the

policy maximizing the agent's *future discounted return* R_t at every timestep

$$R_t = \sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}),$$
$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi,\tau,\sigma_0} \left[R_0\right]. \tag{1}$$

The notation $\mathbb{E}_{\pi,\tau,\sigma_0}$ is used to denote an expectancy 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 $\boldsymbol{s}_0 \sim \sigma_0(\cdot)$ and $\boldsymbol{s}_{t+1} \sim \tau(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$, induced by the environment. Although some RL methods try to directly search for the optimal policy using the objective (1), it is often useful to use (an estimate of) the policy's *action-value function* $Q_\pi(\boldsymbol{s}, \boldsymbol{a})$ for extra guidance

$$Q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,\tau} \left[R_t | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\right].$$

This action-value function satisfies following recursive relationship, known as the Bellman equation

$$Q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,\tau}[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma Q_\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}].$$

A more extensive introduction to the domain of reinforcement learning is given by Sutton & Barto [21]. In this paper we will further limit the discussion to deterministic, off-policy, actor-critic methods, such as 'Deep Deterministic Policy Gradient' (DDPG) [11] and 'Twin Delayed DDPG' (TD3) [5]. These methods consist of two major components: the actor network $\mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$ modelling the deterministic policy (state-action mapping) and the critic network $Q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_Q)$ estimating the optimal state-action value function. Both components are jointly updated, improving one another as training progresses, using experience collected while the agent is interacting with the environment during training. As a deterministic policy maps the same state always to the same action, an external source of stochasticity is often required in order to sufficiently explore the state-action space. Hence, the behavioural policy $\beta(\boldsymbol{s}) = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu) + \epsilon$ with exploration noise $\epsilon \sim N(\boldsymbol{0}, \boldsymbol{\sigma}_{expl})$ is used to collect experience during training instead of the deterministic policy modelled through the actor, making these methods *off-policy*. The collected experience tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ are first stored in a replay buffer $\mathcal{B}$. In a second step, uniformly sampled batches of experience tuples from this buffer are used to update the actor and critic networks.

The critic network is updated by minimizing a squared temporal difference error[3]

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

[3] For TD3 extra twin networks are introduced to avoid overestimation bias and an extra noise term is added to the target policy's actions to smoothen the value estimate [5].

where the primes on the weight vectors denote the usage of target networks to improve the stability of the learning process. The actor network is updated by minimizing the actor loss

$$L_\mu(\boldsymbol{\theta}_\mu) = -\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{B}} \left[Q(\boldsymbol{s}_t, \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu); \boldsymbol{\theta}_Q) \right], \tag{2}$$

leading to an approximation of the deterministic policy gradient [19]

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

3 Motivation

As a first, motivational example, the task of learning a simple overtaking manoeuvre on a three-lane highway is considered (Figure 1). At the start of each episode, the virtual driver (agent) is positioned in the rightmost lane behind a slower lead vehicle, travelling at a constant velocity. The virtual driver can perceive its current velocity components, the relative lateral offset w.r.t. its current lane center and the lanes directly to the left and right, and the relative offset and velocity components w.r.t. the car to overtake. The action space is two-dimensional, consisting of a longitudinal reference velocity and lateral reference offset (which are tracked by lower-level controllers). The reward $r(\boldsymbol{s}_t)$ is a weighted sum of two components $0.75 r_V + 0.25 r_R$. Where r_V is a penalty given when traveling at low velocities — thus rewarding policies which overtake the slow vehicle instead of staying behind it — and r_R is a penalty given when not driving in the rightmost lane (to obtain policies following common rules of the road). More details on the simulation environment and definitions of states and reward signals can be found in Appendix A.

Five policies (each initialized with a different seed) are learned in this environment using the TD3 method [5]. While each of them is able to correctly overtake the slow vehicle, therefore maximizing their long-term reward, only two of them do so in a smooth way. The others suffer from high-frequency oscillations in their lateral reference signals, severely impacting passenger comfort as

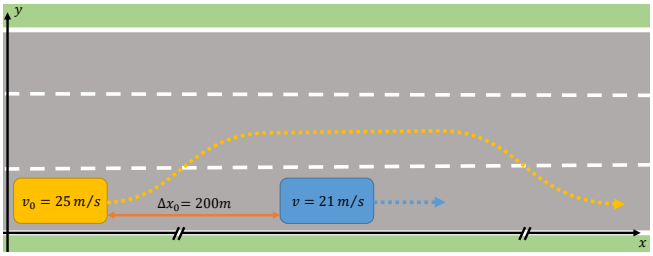

Fig. 1. Schematic overview of the motivational overtaking environment. The virtual driver (agent) is in control of the yellow vehicle and has to overtake the slower moving blue vehicle in front of it.

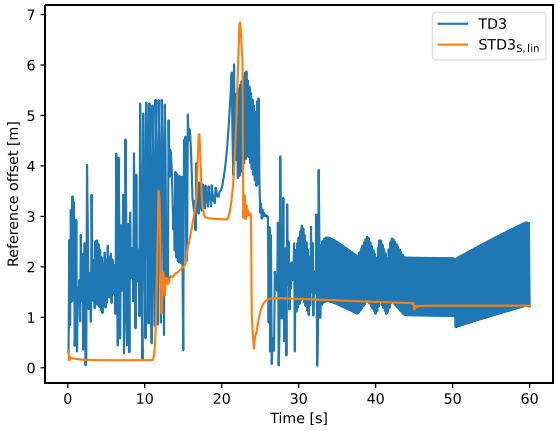

Fig. 2. Lateral reference offset of two policies trained for 250 episodes (with 600 timesteps) on the motivational overtaking environment. The blue line corresponds to the policy trained using the vanilla TD3 method, the orange line corresponds to the policy trained using the smoothed $\text{STD3}_{\text{S,lin}}$ variant (after 50 episodes of smoothing). While both policies correctly learn the overtaking manoeuvre (around $15 - 25s$), the rough reference changes of the TD3 policy prevent its usage in real vehicles.

illustrated in Figure 2. The occurrence of these oscillations throughout training is also quite volatile, as they seem to vanish and reappear within a few training episodes.

For simple environments, such smoothness problems could be dealt with by incorporating extra penalties in the reward signal and/or the usage of derivative control (see Section 1). However, this becomes increasingly more difficult for problems with more complex reward functions or state representations. Without proper care, the resulting policies could take a significant performance hit, as compared to their unconstrained counterparts (see discussion in Section 5.3 and Figure 6). The proposed smoothed TD3 variants in this work are more easily applicable and have higher robustness to such problems.

4 Methodology

To improve the smoothness of the learned policies, different smoothed TD3 (STD3) variants are introduced in the following subsections. First, a brief overview of the used roughness metrics is given.

4.1 Roughness metrics

Different metrics of smoothness or roughness of a curve or control signal exist:

– The integral of the second-order time-derivatives (or its approximation using sums and finite differences for discrete signals), as commonly used in smoothing spline applications [7].

– A metric defined over the frequency spectrum of the time signal, obtained after a Fourier transform, typically used in signal processing [6, 12].
– The average squared temporal difference of consecutive samples [14].

In the context of this paper, we use the third metric and calculate the average roughness of a discrete time signal $\boldsymbol{x}_k, 0 \leq k \leq k_M$ as

$$\bar{\rho} = \frac{1}{k_M} \sum_{k=1}^{k_M} \rho(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k). \tag{3}$$

The *immediate roughness* of the signal is then defined as

$$\rho(\boldsymbol{x}_0, \boldsymbol{x}_1) = \|\boldsymbol{x}_0 - \boldsymbol{x}_1\|_P^2 = (\boldsymbol{x}_0 - \boldsymbol{x}_1)^\top P(\boldsymbol{x}_0 - \boldsymbol{x}_1), \tag{4}$$

where P is a positive definite matrix that can be used to put more or less weight on certain signal components.

Equation (3) can be further generalized to time signals originating from sampling actions from a policy π under an MDP with finite episode length k_M. For the specific case of calculating the average roughness of the sampled actions we have

$$\bar{\rho}_\pi = \mathbb{E}_{\pi,\tau,\sigma_0} \left[\frac{1}{k_M} \sum_{t=1}^{k_M} \rho(\boldsymbol{a}_{t-1}, \boldsymbol{a}_t) \right]. \tag{5}$$

Different ways to approximate this expectancy will be given in the next subsection. Beside being a simple metric to calculate, this definition of roughness will turn out to be advantageous when combined with model-free reinforcement learning schemes.

Notice that taking $P = I$ gives the *unscaled roughness*, using the Euclidean norm of the action difference. Another commonly used choice for P throughout this paper is the diagonal matrix with elements $p_{i,i} = \Delta a_i^{-2}$ where Δa_i is the maximum absolute difference between the i-th component of two actions. This gives rise to the *immediate normalized roughness* ρ_{norm}, which is less impacted by action components with a larger range of possible values (i.e. with a higher Δa_i).

4.2 Smoothed TD3

We propose to modify the actor loss (2) by adding an extra weighted smoothness term, following existing smoothness regularization techniques for neural networks [6]. In this case, the smoothness term approximates the average roughness of the policy

$$R_{\text{smooth}} \approx \bar{\rho}_\pi.$$

The actor weights are then updated by minimizing the total loss

$$L_{\mu,\text{smooth}} = L_\mu + \lambda_s R_{\text{smooth}}. \tag{6}$$

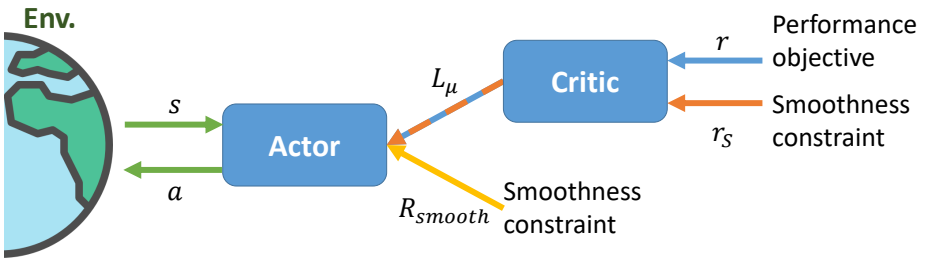

Fig. 3. Schematic overview comparing the information flow of the applied smoothness constraints using reward penalties (orange) and actor regularization (yellow). With reward penalties, the information flow is indirect, i.e. it is first used to update the critic model and this updated critic model is then used to update the actor model. With regularization, the information flow is directly acting on the actor model, interfacing with the environment.

This effectively forces the actor network to not only optimize the expected discounted value, but also to force its outputs corresponding to consecutive states ($\boldsymbol{s}_t$ and $\boldsymbol{s}_{t+1}$) to be similar in the chosen roughness norm (4).

Figure 3 provides a schematic overview showing the major difference between smoothness constraints imposed through the reward and those imposed through actor regularization. Notice the indirect application of the smoothness constraints on the actor model, through the critic model, when using the reward signal. As a consequence, the actor model can not be smoothed properly whenever the critic is not able to accurately capture the underlying smoothness constraints. In fact, as the critic is only an approximation of the optimal value function, there is no guarantee that the critic is able to capture this relationship at all. In practice this means many experience samples are required in order for the critic to discover the complex relationship between states, actions and accumulated returns — encoding both the performance objective and the smoothness constraints — without any certainty of success. Hence, while such an indirect information flow works reasonably well for complex functions of states and actions, such as the performance objective (1), it is needlessly complex for the applied smoothness constraints, which only depend on consecutive transitions in state-action space. In contrast, when using the regularization term, the smoothness constraints are applied directly on the actor model, without any intermediate approximation step. Moreover, in the calculation of the regularization term we can explicitly utilize the tight temporal connection in state-action space of the smoothness constraints, leading to a more sample-efficient smoothing effect.

The introduced hyperparameter $\lambda_s \geq 0$ in (6) can be tuned to trade-off performance and smoothness objectives. Low values will result in policies optimizing their future rewards, but they might be non-smooth (see the example of Section 3). High values will result in smooth policies, but might not always achieve a good performance. Remark that the effective amount of smoothing also depends on the definition of the reward signal. After all, the smoothness

weight λ_s balances the regularization term R_{smooth} *relatively* against the actor loss L_μ (6), which is proportional to the average Q-value (2). Hence, the same value of λ_s will have less smoothing impact on environments with larger rewards (in absolute value), leading to Q-values and actor loss values with higher order of magnitude, effectively suppressing the smoothness regularization term. For this reason, it is recommended to normalize the reward signal, prior to storing it in the replay buffer. Then, bounds on the L_μ term can be calculated and traded off against the maximum value of R_{smooth}, which is bounded by $\max_{i,j} \rho(\boldsymbol{a}_i, \boldsymbol{a}_j)$. Such reward normalization is also applied in the conducted experiments of Section 5 and shows the improved robustness of the smoothness weight parameter: a single λ_s value leading to acceptable policy smoothing across varying environments.

Different approximations of the used roughness metric $\bar{\rho}_\pi$ (5) and different schedules for the smoothness weight λ_s lead to different STD3 variants. In the remainder of this subsection, these different variants will be introduced. Remark that the specific combination $\text{STD3}_{\text{C,fix}}$ corresponds to the temporal smoothness regularization term of the CAPS method presented by Mysore et al. [12].

Roughness approximation The expectancy in (5) can be approximated in different ways, leading to two STD3 variants introduced below. Both approximations can reuse the same batch of sampled experience from the replay buffer $\mathcal{B}$, used to calculate (2). Hence, the extra smoothness regularization term can be easily plugged into existing training loops of off-policy actor-critic methods such as DDPG, TD3, Proximal Policy Optimization (PPO) [18] and Soft Actor Critic (SAC) [9].

The first *supervised smoothing* ($\text{STD3}_{\text{S},\bullet}$) variant uses the current action $\boldsymbol{a}_t$, as sampled from the replay buffer, and the next action $\tilde{\boldsymbol{a}}_{t+1} = \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)$, as given by the policy for the next state $\boldsymbol{s}_{t+1}$, in the regularizer calculation

$$R_{\text{smooth}}(\boldsymbol{\theta}_\mu) = \mathbb{E}_{(\boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \sim \mathcal{B}} \left[\rho(\boldsymbol{a}_t, \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)) \right].$$

The name 'supervised' stems from the fact that the resulting smoothness regularizer forces actor network outputs to be similar to given targets (the sampled $\boldsymbol{a}_t$ actions from the replay buffer) in the chosen roughness norm (4), analogous to the classical supervised learning setting.

The second *contrastive smoothing* ($\text{STD3}_{\text{C},\bullet}$) variant uses the current action $\tilde{\boldsymbol{a}}_t = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$ and next action $\tilde{\boldsymbol{a}}_{t+1} = \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)$, both as given by the policy for the current and next states, in the regularizer calculation

$$R_{\text{smooth}}(\boldsymbol{\theta}_\mu) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}) \sim \mathcal{B}} \left[\rho(\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu), \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)) \right].$$

In this case the name 'contrastive' stems from the fact that two outputs from the same actor network are forced to be similar in the chosen roughness norm (4), comparable to the contrastive learning setting. Remark that the calculation of $\tilde{\boldsymbol{a}}_t = \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu)$ and the corresponding forward and backward pass through the actor network are already performed for the calculation of the actor loss L_μ (2).

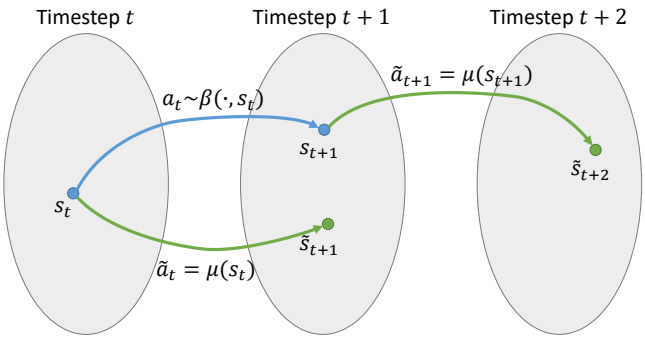

Fig. 4. Comparison of supervised and contrastive smoothing approximations. In blue are the states and actions present in the replay buffer $\mathcal{B}$. In green are the actions calculated during evaluation of the smoothness regularizer. For supervised smoothing, $\rho(\boldsymbol{a}_t, \tilde{\boldsymbol{a}}_{t+1})$ is used for the immediate roughness approximation. For contrastive smoothing, $\rho(\tilde{\boldsymbol{a}}_t, \tilde{\boldsymbol{a}}_{t+1})$ is used instead.

Hence, there is no significant computational overhead for using the contrastive variant, as compared to the supervised variant.

Notice that only a single $(\boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ or $(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$ experience sample is needed to already start improving the smoothness of the actor model for the sampled state transition. As previously mentioned this is more sample-efficient than the usage of smoothness penalties in the reward, which require multiple experience samples to uncover the underlying smoothness goal.

A comparison of the used state and action information by both variants is shown in Figure 4. The supervised variant has the strongest temporal connection between the consecutive actions $\boldsymbol{a}_t$ and $\tilde{\boldsymbol{a}}_{t+1}$ used in the regularizer calculation. More precisely, it is guaranteed that taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ can lead to state $\boldsymbol{s}_{t+1}$, where the current deterministic policy will take action $\tilde{\boldsymbol{a}}_{t+1}$. Hence, forcing the actor output $\mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}_\mu)$ to be similar to $\boldsymbol{a}_t$ in the chosen roughness norm, will indeed improve the temporal smoothness of the policy. For the contrastive variant there is no such strong temporal connection, as taking action $\tilde{\boldsymbol{a}}_t$ in state $\boldsymbol{s}_t$ does not necessarily lead to state $\boldsymbol{s}_{t+1}$. This discrepancy (between states $\boldsymbol{s}_{t+1}$ and $\tilde{\boldsymbol{s}}_{t+1}$) will however diminish as training goes on and the behavioural policy β becomes more similar to the deterministic policy μ.

Smoothing schedules A smoothing schedule is a function $\lambda_s(f) : [0\,;1] \rightarrow [\lambda_{s,m}\,;\lambda_{s,M}]$ mapping the current training progress $f = \frac{T}{T_M}$ to a value for the smoothing weight λ_s, where the current training episode is denoted by T and the total amount of training episodes by T_M. All considered schedules are bounded, i.e., $0 \leq \lambda_{s,m} \leq \lambda_s(f) \leq \lambda_{s,M} \quad \forall f \in [0\,;1]$.

The most straightforward STD3$_{\bullet,\text{fix}}$ variant keeps the smoothness weight λ_s fixed during the whole training process. The smoothness weight then becomes another hyperparameter to tune, depending on the complexity of the environ-

ment and desired amount of policy smoothing

$$\lambda_s(f) = \lambda_s = \lambda_{s,m} = \lambda_{s,M} \quad \forall f \in [0\,;1].$$

On difficult environments, it can be hard to find good smoothness weight values under the fixed scheme. Extra smoothing constraints too early in the training process can hamper the optimization process, leading to suboptimal learned policies. The second STD3$_{\bullet,\text{lin}}$ variant tries to resolve this issue by splitting the training process in two phases: an initial phase from $f = 0$ to $f = f_{p1} < 1$, followed by a second phase until $f = 1$. In the first training phase λ_s remains equal to $\lambda_{s,m} = 0$, allowing the agent to maximally optimize the policy's performance without any smoothness constraints. Hence, during this phase, there is no difference between the smoothed and raw TD3 method. In the second training phase, after a reasonably good policy has been found, the smoothness constraints are gradually introduced by linearly increasing λ_s as training progresses (until the end of training). The resulting schedule effectively 'smoothens out' the policies obtained after the first training phase

$$\lambda_s(f) = \begin{cases} 0 & 0 \leq f \leq f_{p1} \quad \textit{(Phase 1)} \\ \lambda_{s,M} \dfrac{f - f_{p1}}{1 - f_{p1}} & f_{p1} < f \leq 1 \quad \textit{(Phase 2)} \end{cases}.$$

In practice, the determination of f_{p1} — the end of phase 1 — might require some trial-and-error experiments. Moreover, it might not always be necessary to maximally reduce the policy's roughness at the potential cost of a reduced performance. For some applications, keeping the roughness below a certain threshold may satisfy all real-world requirements on smoothness. The last STD3$_{\bullet,\text{adapt}}$ variant addresses both issues, by automatically putting more weight on smoothing or value optimization depending on a current roughness estimate $\tilde{\rho}_\pi$. This roughness estimate tries to approximate $\bar{\rho}_\pi$ (5) by averaging the measured average roughness (3) over E evaluation episodes (i.e. using the deterministic policy $\boldsymbol{a} = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$ without exploration noise)

$$\tilde{\rho}_\pi = \frac{1}{E} \sum_{e=1}^{E} \frac{1}{N_e} \sum_{t=1}^{N_e} \rho(\boldsymbol{a}_{t-1}^e, \boldsymbol{a}_t^e), \tag{7}$$

where the superscript on the actions denotes the specific evaluation episode in which they occurred. This estimate is then reevaluated every T_e training episodes, keeping the smoothness weight constant in between the evaluations

$$\begin{aligned} \lambda_s(f) &= \lambda_{s,k} \quad \forall f : kT_e \leq fT_M < (k+1)T_e, \\ \lambda_{s,0} &\in [\lambda_{s,m}\,; \lambda_{s,M}], \\ \lambda_{s,k+1} &= \begin{cases} \max\{\lambda_{s,m}, s^- \lambda_{s,k}\} & \tilde{\rho}_\pi < \rho_m \\ \lambda_{s,k} & \tilde{\rho}_\pi \in [\rho_m\,; \rho_M] \\ \min\{\lambda_{s,M}, s^+ \lambda_{s,k}\} & \tilde{\rho}_\pi > \rho_M \end{cases}. \end{aligned}$$

The resulting smoothing schedule is piecewise constant and adapts the amount of smoothing throughout training, based on the specific needs. More precisely, in case the roughness estimate lies above a predefined upper threshold ρ_M, more weight is put on the policy smoothing for the next T_e training episodes — through multiplication of λ_s by $s^+ > 1$. Similarly, in case the roughness estimate lies below a predefined lower threshold ρ_m, more weight is put (again) on policy optimization for the next T_e training episodes — through multiplication of λ_s by $s^- \in (0\,;1)$.

This last variant could be seen as an ad-hoc strategy to find an approximate solution of the constrained MDP (CMDP)

$$\begin{aligned} \pi^* = \arg\max_{\pi} & \mathbb{E}_{\pi,\tau,\sigma_0}\left[R_0\right] \\ \text{s.t. } & \bar{\rho}_\pi \leq \rho_M. \end{aligned}$$

Here, the focus is not to maximally reduce the roughness of the obtained policies (as is the case for the first two smoothing schedules), but rather to reduce the roughness of the policies to an acceptable level determined by the specified roughness thresholds. In optimal control, such a constraint is also referred to as a *slew rate constraint*.

5 Experimental results

Three different experiments were conducted to compare the different smoothed TD3 variants across different environments of varying complexity. The training and evaluation procedures are briefly described first.

For every hyperparameter configuration, the experiment is repeated five times, using five different seeds for initialization. After every T_e training episodes or k_e training timesteps, E independent evaluation episodes are executed to get an estimate of the learned policy's average performance $\tilde{R}_\pi$ and roughness $\tilde{\rho}_\pi$ (7)

$$\tilde{R}_\pi = \frac{1}{E}\sum_{e=1}^{E}\frac{1}{N_e}\sum_{t=1}^{N_e} r(\boldsymbol{s}_{t-1}^e, \boldsymbol{a}_{t-1}^e, \boldsymbol{s}_t^e).$$

To summarize these average evaluation metrics and make a comparison between different settings easier, only evaluation metrics of the best B episodes, occurring in the last T_B episodes of the training process[4] are retained. Note that the best B episodes are determined based on the average evaluation performance $\tilde{R}_\pi$ only. Combined with the five independent repeats of each experiment, this leads to $5B$ datapoints, from which comparison statistics are calculated (e.g. a mean value and standard deviation).

[4] This is to guarantee proper convergence of both the performance and smoothness objective on every environment.

5.1 Highway overtaking

As the first experiment, let us quickly revisit the motivational example of Section 3, where the goal was to learn a policy that can overtake a slow vehicle in the rightmost lane of a highway (Figure 1). First, five policies (initialized using different seeds) were trained using the TD3 method without smoothness constraints. Afterwards, five policies (reusing the same five seeds) were trained using the $\text{STD3}_{\text{S,lin}}$ method to smoothen out the previously obtained policies after an initial 200 training episodes ($f_{p1} = 2/3$). While almost all TD3 policies suffered from jerky actions throughout training, the smoothness has improved a lot for the STD3 policies. Only one policy still had some oscillatory reference actions after smoothing, but for only a fraction of the time as compared to the unsmoothed policies. Figure 2 shows an evaluation episode of one of the five policies after 250 training episodes (i.e. after 50 smoothing episodes for the STD3 policy).

5.2 OpenAI benchmarks

In this second experiment, the different STD3 variants will be compared against each other (and the standard TD3 method) on 10 commonly used OpenAI gym environments[5]. We use a customized version of the latest Stable-Baselines implementation[6] to perform these experiments. Their tuned hyperparameters for the TD3 algorithm are reused with a few exceptions for some environments requiring a longer training time. A summary of the used hyperparameters and full environment names can be found in Appendix B. The used smoothness parameters are summarized in Table 1. Notice that for the fixed and linear smoothing schedule, the same parameters could be reused across all environments. This was possible due to the normalization of states and rewards, prior to storage in the replay buffer, and shows the robustness of these smoothness parameters. For the adaptive smoothing schedule, the extra imposed smoothness constraint was set as to reduce the roughness of the policies by half, as compared to standard TD3. More precisely, we put the maximum threshold ρ_M equal to approximately half the roughness of policies obtained using default TD3. The minimum threshold was set to roughly 90% of the maximum threshold's value.

The training and evaluation procedure outlined at the beginning of this section was followed using five independent repeats with $E = 5$ evaluation episodes every $k_e = 5000$ training timesteps. Evaluation metrics of the five best episodes occurring in the last 20% of the training process were used to make the comparison between different configurations ($B = 5, T_B = 0.2T_M$). The mean values and standard deviations for the performance and roughness metrics are summarized in Table 2.

In general, all investigated STD3 variants significantly improve the policy smoothness. However, some do so at the cost of a reduced performance. For example, the $\text{STD3}_{\text{S,fix}}$ method consistently leads to the smoothest policies, but

[5] https://gym.openai.com
[6] https://github.com/DLR-RM/stable-baselines3

Table 1. Smoothness parameters for the different environments. The normalized roughness norm ρ_{norm} was used for both the regularizer calculation R_{smooth} and roughness estimation $\tilde{\rho}_\pi$. The remaining parameters for the adaptive variant were set as follows for every environment: $\lambda_{s,0} = 1 \cdot 10^{-4}$, $\lambda_{s,m} = 1 \cdot 10^{-6}$, $\lambda_{s,M} = 1$.

Environment	**STD3$_{\bullet,\text{fix}}$**	**STD3$_{\bullet,\text{lin}}$**		**STD3$_{\bullet,\text{adapt}}$**	
	λ_s	f_{p1}	$\lambda_{s,M}$	ρ_m	ρ_M
Ant	0.2	0.6	0.4	0.36	0.4
Bipedal	0.2	0.6	0.4	0.16	0.18
Hopper	0.2	0.6	0.4	0.04	0.05
IDP	0.2	0.6	0.4	0.08	0.09
IPS	0.2	0.6	0.4	0.08	0.1
Lunar	0.2	0.6	0.4	0.18	0.2
Minitaur	0.2	0.6	0.4	0.36	0.4
MCC	0.2	0.6	0.4	0.004	0.005
Pendulum	0.2	0.6	0.4	0.08	0.1
Walker	0.2	0.6	0.4	0.16	0.18

often not to the best performing ones. The reverse situation is also observable for the STD3$_{\text{C,fix}}$ method: this method leads to the best performing policies, but other methods can typically reduce the roughness slightly more. It should be noted however that results lie close together for some environments. Furthermore, the addition of extra smoothness constraints does not always lead to a reduction in performance. On five environments the best performing TD3 policies are outperformed by an STD3 variant. In particular, the STD3$_{\text{C,lin}}$ method seems to find the best balance between performance and smoothness, as it leads most often (on five environments) to both the best performing and smoothest policies.

It might not always be required to obtain the absolute best performing or smoothest policy though. Depending on the performance-smoothness trade-off acceptable for a given application, the best STD3 variant can be selected. The maximum flexibility in defining the desired smoothing behaviour is obtained using the adaptive smoothing schedules. As can be seen from Table 2, reducing the roughness by 50% seems to succeed in at least 6 different environments. This comes without large performance costs, as we still obtain best performing policies in half of the environments. The adaptive smoothing schedules seem to have the most difficulty on environments where the best performing TD3 policies have a high smoothness variability (high roughness variance in Table 2). A possible explanation for this behaviour is the observed abrupt vanishing and reappearance of rough actions throughout training. This might lower the smoothness weight, even though the policies are still 'vulnerable' to emerging jerky actions. One possible solution for this, is the usage of an exponential moving average for the roughness estimate, instead of recalculating it from scratch every evaluation. In

Table 2. Benchmark results of the different STD3 variants. All values are relative changes with respect to the mean value obtained by the TD3 algorithm. The best performing policies (top) and smoothest policies (bottom) are highlighted in bold. The bottom row summarizes the data, denoting the number of environments for which *both* the best performing policy *and* smoothest policy was obtained using the given method. For the adaptive variants, the number in parentheses is the amount of environments for which the best performing policy was obtained, under the given smoothness constraint.

Environment	**Average return (↑) [%]**						
	TD3	$\text{STD3}_{\text{S,fix}}$	$\text{STD3}_{\text{S,lin}}$	$\text{STD3}_{\text{S,adapt}}$	$\text{STD3}_{\text{C,fix}}$	$\text{STD3}_{\text{C,lin}}$	$\text{STD3}_{\text{C,adapt}}$
Ant	**0.00 ± 4.92**	−21.87 ± 5.33	−11.47 ± 7.29	−20.26 ± 38.03	**−4.00 ± 8.68**	**−1.62 ± 6.29**	**−1.19 ± 3.62**
Bipedal	0.00 ± 0.88	−3.45 ± 1.02	−0.24 ± 1.24	−0.09 ± 2.09	**2.95 ± 0.61**	**2.99 ± 0.56**	1.74 ± 0.39
Hopper	0.00 ± 4.18	−4.92 ± 5.76	−0.86 ± 4.47	−2.25 ± 3.50	**9.70 ± 2.60**	−2.48 ± 4.11	2.72 ± 5.74
IDP	**0.00 ± 0.02**	−0.06 ± 0.01	−0.03 ± 0.02	**−0.02 ± 0.04**	−0.02 ± 0.03	**−0.01 ± 0.02**	**−0.02 ± 0.02**
IPS	**0.00 ± 0.09**	−2.58 ± 2.65	−0.95 ± 0.46	**−0.04 ± 0.07**	**−0.08 ± 0.21**	−0.19 ± 0.24	−0.15 ± 0.17
Lunar	0.00 ± 2.88	**5.02 ± 1.75**	1.60 ± 4.49	1.51 ± 2.10	**5.91 ± 4.24**	1.94 ± 3.11	2.30 ± 2.70
Minitaur	0.00 ± 15.42	−27.89 ± 32.24	−23.32 ± 22.35	3.15 ± 27.48	**25.84 ± 9.73**	−3.03 ± 26.09	**19.75 ± 6.22**
MCC	0.00 ± 0.15	−19.21 ± 40.61	**0.72 ± 0.23**	0.24 ± 0.11	0.36 ± 0.10	0.35 ± 0.09	−20.10 ± 40.00
Pendulum	**0.00 ± 26.96**	**−3.35 ± 28.93**	**−4.37 ± 28.59**	**−1.40 ± 29.14**	**−1.52 ± 26.10**	**−0.36 ± 26.57**	**−2.31 ± 26.73**
Walker	**0.00 ± 4.55**	**−2.16 ± 2.94**	**1.02 ± 3.21**	**−1.60 ± 6.16**	**2.18 ± 5.32**	**−1.78 ± 3.80**	**−1.82 ± 3.44**

Environment	**Average roughness (↓) [%]**						
	TD3	$\text{STD3}_{\text{S,fix}}$	$\text{STD3}_{\text{S,lin}}$	$\text{STD3}_{\text{S,adapt}}$	$\text{STD3}_{\text{C,fix}}$	$\text{STD3}_{\text{C,lin}}$	$\text{STD3}_{\text{C,adapt}}$
Ant	0.00 ± 9.89	−83.93 ± 1.02	−83.46 ± 1.97	−63.53 ± 18.81	−84.62 ± 1.91	**−87.04 ± 1.65**	−55.17 ± 5.64
Bipedal	0.00 ± 7.61	**−85.31 ± 2.40**	**−84.45 ± 3.05**	−48.76 ± 4.46	−82.08 ± 4.78	**−85.59 ± 2.54**	−53.15 ± 5.64
Hopper	0.00 ± 27.74	**−70.23 ± 4.62**	−65.40 ± 8.73	−43.37 ± 8.60	−66.12 ± 8.95	**−75.98 ± 8.93**	−48.31 ± 7.49
IDP	0.00 ± 58.45	**−97.82 ± 0.69**	**−97.31 ± 1.10**	−16.07 ± 88.20	**−98.66 ± 2.29**	**−98.92 ± 1.70**	−33.76 ± 42.98
IPS	0.00 ± 146.19	**−99.70 ± 0.22**	−98.61 ± 0.64	−0.27 ± 139.53	**−99.74 ± 0.20**	−99.38 ± 0.40	−68.86 ± 36.79
Lunar	0.00 ± 32.70	**−96.79 ± 1.11**	**−96.96 ± 1.26**	−48.66 ± 18.96	−91.13 ± 2.44	−92.04 ± 3.19	−48.00 ± 25.37
Minitaur	0.00 ± 16.36	**−82.00 ± 5.70**	−76.02 ± 6.79	−49.75 ± 4.14	−59.03 ± 4.63	−66.41 ± 11.81	−52.63 ± 4.38
MCC	0.00 ± 63.12	**−74.66 ± 14.05**	**−71.53 ± 5.11**	−33.44 ± 24.95	**−68.87 ± 2.37**	**−69.75 ± 4.27**	−20.68 ± 69.77
Pendulum	0.00 ± 114.96	**−98.81 ± 0.51**	**−98.75 ± 0.51**	−38.01 ± 82.90	−98.17 ± 1.22	**−98.74 ± 0.74**	13.20 ± 125.25
Walker	0.00 ± 18.60	−77.15 ± 2.55	−77.42 ± 3.22	−48.39 ± 9.01	−76.94 ± 4.24	**−82.79 ± 4.38**	−52.17 ± 11.10
# Best	–	2	2	– (4)	1	**5**	– (5)

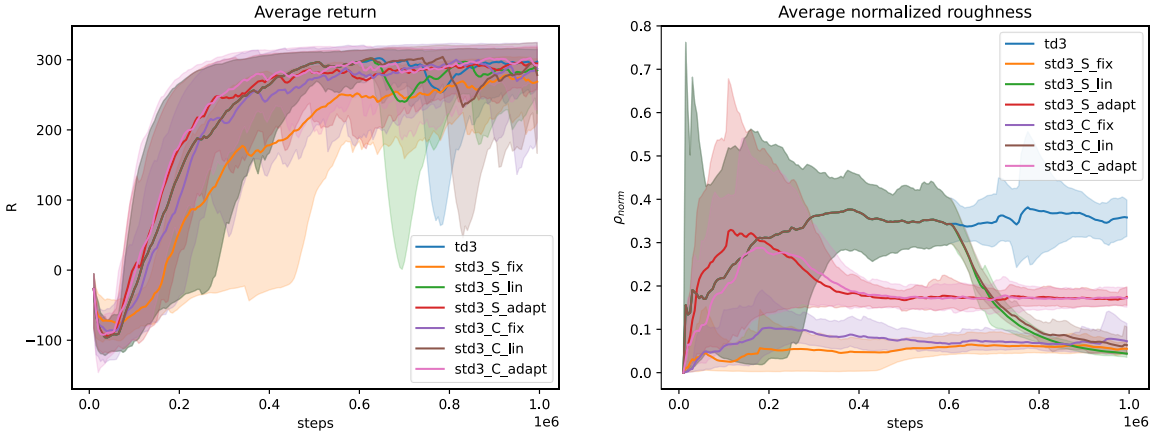

Fig. 5. Exponentially smoothed ($\alpha = 0.1$) evolution of the average return (left) and roughness (right) of the policies as training goes on for the Bipedal environment.

this way, past non-smooth behaviour is accounted for longer. It is left for future work to investigate such more elaborate smoothing schemes.

Finally, Figure 5 shows the different smoothing schedules' effect on the policy's roughness throughout training. The simplest fixed scheme immediately acts on the policy, from the very start of the training process, giving no chance for policies to become too rough. In the linear scheme, the same roughness behaviour as for the TD3 method is obtained during the initial training phase. As soon as the smoothing phase starts, the roughness is drastically reduced to roughly the same level as the fixed scheme. Both of these clearly try to reduce the roughness as much as possible. The final adaptive scheme starts to smoothen out the policies as soon as the predefined threshold is crossed, after which the roughness settles around this threshold value.

5.3 Highway driving

In this last experiment, we investigate the best performing $\text{STD3}_{\text{C,lin}}$ variant and compare it against its supervised counterpart on a more complex environment. The same simulator as in the first experiment is used, but this time there are multiple moving vehicles on the three-lane highway. The objective in this environment is to travel as fast as possible, while respecting all traffic rules (speed limit, keep right) and safety constraints (preventing crashes). Details can be found in Appendix A.

Once again, all configurations are repeated five times with differently seeded initializations. Average performance and smoothness metrics are calculated from one evaluation episode after every training episode ($E = T_e = 1$). The smoothness estimate is calculated as

$$\tilde{s}_\pi = \frac{1}{N_e} \sum_{t=1}^{N_e} \big(\exp[-\rho(\boldsymbol{a}_{t-1}, \boldsymbol{a}_t)] - 1 \big),$$

giving values closer to 0 for smoother policies and values closer to -1 for non-smooth policies. The performance metric is the accumulated sum of normalized rewards *without smoothness penalty* r_S, with maximum value 0 and minimum value -1. All policies were trained for $T_M = 300$ episodes with $k_M = 5000$ timesteps, smoothing started after one third of the training was done ($f_{p1} = 1/3$ empirically determined) using different values of the final (maximum) smoothing weight $\lambda_{s,M}$ and of the smoothness penalty weight w_S in the reward. Performance and smoothness statistics are calculated from the best 20 episodes occurring in the second half of the training process ($B = 20$, $T_B = 0.5T_M$). A summary of the results is shown in Figure 6. The first experiment (on top) compares the $\text{STD3}_{\text{S,lin}}$ method with the standard TD3 method using smoothness penalties in the reward. Clearly, the smoothness of the obtained policies is increased for both approaches. However, using smoothness penalties quickly becomes impractical, as performance starts to deteriorate for increasing values of w_S. Using smoothed TD3 on the other hand, results in policies having higher smoothness values without any performance reduction. Naturally, this only holds up to certain

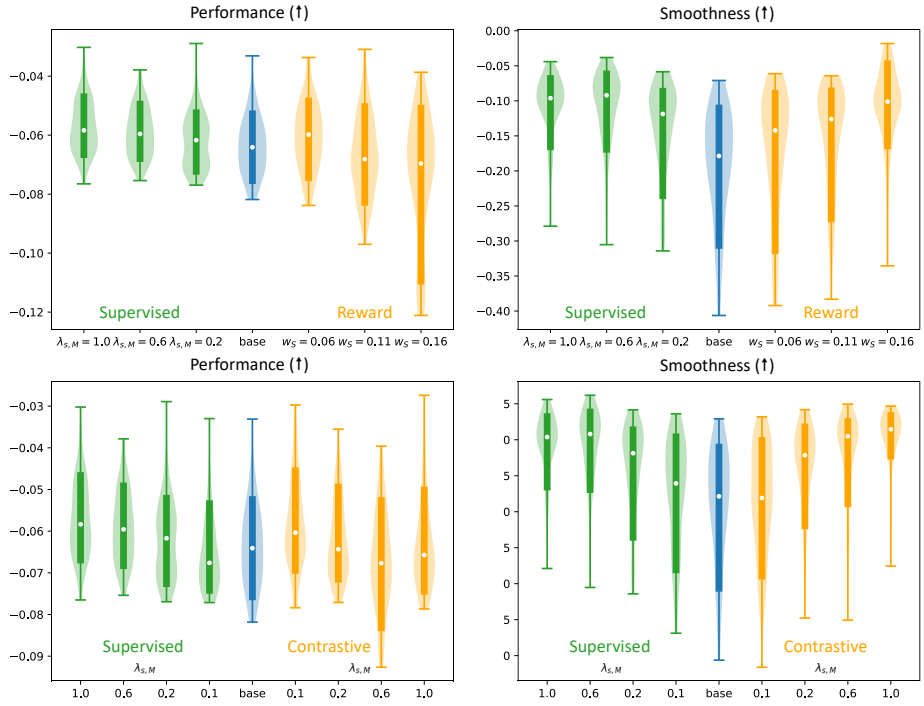

Fig. 6. Comparison of performance (left) and smoothness (right) for policies trained on the highway driving environment. In blue the standard TD3 method without any smoothness constraints or penalties. In green the supervised $\text{STD3}_{\text{S,lin}}$ method. In orange the policies trained with extra smoothness penalties in the reward signal (top) or using the contrastive $\text{STD3}_{\text{C,lin}}$ method (bottom). The whiskers denote the minimum/maximum values, the shaded area shows an estimate of the underlying distribution, the middle rectangle spans from the first to the third quartile and the white dot shows the mean value.

limits but policies trained using STD3 were found to be much more robust to such performance declines empirically. Hence less time can be spent on finetuning the trade-off between performance and smoothness, which typically required trying to fit in smoothness penalties into an already existing reward signal.

In the second experiment (bottom of Figure 6), the constrastive and supervised STD3 variants were compared (both using the 'lin' smoothing schedule). Both lead to roughly the same amount of smoothness improvement for different values of $\lambda_{s,M}$. Performance stays roughly at the same level, although there is a slight increase for the supervised variant and a slight decrease for the contrastive variant. This might be a bit surprising, as the results on the openAI gym environments seemed to indicate the contrastive variants had superior performance. But this confirms the fact that different environments require different smoothing measures. For the simplest environments, an extra penalty in the reward might suffice. As complexity increases, the smoothed TD3 variants become necessary to prevent severe performance deterioration. Finally, for the most complex environments (such as chaotic systems [2]), it seems the stronger temporal connection of actions in the supervised smoothing setting, makes them more relevant. In such environments initial policy estimates might be far off from the later, more optimal policies; and slight changes in the chosen actions could lead to vastly different state transitions. Both contributing to higher discrepancies in the compared states of the contrastive smoothing method (see Figure 4).

6 Conclusion

In this paper we introduced different smoothed TD3 (STD3) variants to improve the learned policy's temporal smoothness. The specific choice of roughness metric (5) used for the calculation of both the smoothness regularization term and the smoothness estimate, makes it easily combinable with existing off-policy, policy-based and actor-critic reinforcement learning algorithms. Experiments using normalized returns and roughness metrics show that the extra smoothness weight hyperparameter generalizes well across a variety of different environments, leading to smooth policies without significant performance deterioration. For more fine-grained control over the desired smoothness–performance trade-off, a proper smoothing schedule can be selected. From these schedules, the adaptive smoothing variant is the most versatile. Using an estimate of the currently learned policy's roughness on evaluation episodes, it tries to automatically reduce this policy's roughness below a predefined threshold set at the start of training. The resulting policy is an approximate solution of the constrained MDP with added smoothness constraints.

A possible path forward is the application of the introduced smoothing regularizers to other actor-critic methods, such as PPO and SAC. Although a similar investigation by Mysore et al. [12] observed smoothness regularization to be mostly effective for TD3 as "*soft-policies such as PPO and SAC appear to learn relatively smoother policies on their own*". Another direction of future work can be the investigation of other methods to deal with constrained MDPs, such as

Constrained Policy Optimization (CPO) [1] or Lagrangian methods [20], and compare them with the adaptive STD3 variant introduced here.

Acknowledgements

The presented results were obtained under Ford Alliance Project KUL0076, funded by Ford.

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] Bucci, M.A., Semeraro, O., Allauzen, A., Wisniewski, G., Cordier, L., Mathelin, L.: Control of chaotic systems by deep reinforcement learning. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **475**(2231), 20190351 (2019). https://doi.org/10.1098/rspa.2019.0351

[3] Chen, W., Xiao, H., Wang, Q., Zhao, L., Zhu, M.: Lateral Vehicle Dynamics and Control. John Wiley & Sons, Ltd, 2nd edn. (2016). https://doi.org/10.1002/9781118380000

[4] Chisari, E., Liniger, A., Rupenyan, A., Van Gool, L., Lygeros, J.: Learning from Simulation, Racing in Reality. arXiv preprint 2011.13332 [cs.RO] (2020)

[5] 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)

[6] Girosi, F., Jones, M., Poggio, T.: Regularization Theory and Neural Networks Architectures. Neural Computation **7**(2), 219–269 (1995). https://doi.org/10.1162/neco.1995.7.2.219

[7] Green, P., Silverman, B.W.: Nonparametric Regression and Generalized Linear Models. Chapman and Hall/CRC (may 1993). https://doi.org/10.1201/b15710

[8] Ha, S., Xu, P., Tan, Z., Levine, S., Tan, J.: Learning to Walk in the Real World with Minimal Human Effort. arXiv preprint 2002.08550 [cs.RO] (2020)

[9] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: 35th International Conference on Machine Learning, ICML 2018. vol. 5, pp. 2976–2989 (2018)

[10] Kesting, A., Treiber, M., Helbing, D.: General lane-changing model MOBIL for car-following models. Transportation Research Record pp. 86–94 (2007). https://doi.org/10.3141/1999-10

[11] 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)

[12] Mysore, S., Mabsout, B., Mancuso, R., Saenko, K.: Regularizing Action Policies for Smooth Control with Reinforcement Learning. arXiv preprint 2012.06644 [cs.RO] (2020)

[13] Nageshrao, S., Tseng, H.E., Filev, D.: Autonomous highway driving using deep reinforcement learning. Conference Proceedings - IEEE International

Conference on Systems, Man and Cybernetics pp. 2326–2331 (mar 2019). https://doi.org/10.1109/SMC.2019.8914621

[14] Raffin, A., Kober, J., Stulp, F.: Smooth Exploration for Robotic Reinforcement Learning. arXiv preprint 2005.05719 [cs.LG] (2020)

[15] Rodriguez-Ramos, A., Sampedro, C., Bavle, H., de la Puente, P., Campoy, P.: A Deep Reinforcement Learning Strategy for UAV Autonomous Landing on a Moving Platform. Journal of Intelligent and Robotic Systems: Theory and Applications **93**(1-2), 351–366 (2019). https://doi.org/10.1007/s10846-018-0891-8

[16] Rosca, M., Weber, T., Gretton, A., Mohamed, S.: A case for new neural network smoothness constraints. arXiv preprint 2012.07969 [stat.ML] (2020)

[17] Saxena, D.M., Bae, S., Nakhaei, A., Fujimura, K., Likhachev, M.: Driving in Dense Traffic with Model-Free Reinforcement Learning. In: Proceedings - IEEE International Conference on Robotics and Automation. pp. 5385–5392 (sep 2020). https://doi.org/10.1109/ICRA40945.2020.9197132

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv preprint 1707.06347 [cs.LG] (2017)

[19] 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)

[20] Stooke, A., Achiam, J., Abbeel, P.: Responsive safety in reinforcement learning by PID Lagrangian Methods. arXiv preprint 2007.03964 [math.OC] pp. 9070–9080 (2020)

[21] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, A Bradford book, vol. 258. MIT Press, 1st edn. (1998)

[22] 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

[23] Wang, P., Chan, C.Y., De La Fortelle, A.: A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers. In: IEEE Intelligent Vehicles Symposium, Proceedings. vol. 2018-June, pp. 1379–1384 (oct 2018). https://doi.org/10.1109/IVS.2018.8500556

A Autonomous highway driving environment

The results shown in Section 5 for the highway overtaking and driving environments are obtained using a proprietary highway simulator. In this section the most relevant components of this simulator will be briefly discussed. See also Figure 1 for a schematic overview of the overtaking environment.

A.1 Roads

All experiments were conducted on a three lane highway. For the overtaking environment, this highway was straight along the whole trajectory. For the driving environment, the highway was 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.

A.2 Vehicles

Every vehicle in the simulator follows the kinematic bicycle model (KBM) [3] to update its state based on the selected inputs

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

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 low level 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 and desired lateral position, 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 all 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 gaps and velocities w.r.t. neighbouring traffic. Internally, the simulator discretizes time with step size $\Delta t = 0.1s$ and a Runge-Kutta integration scheme to calculate subsequent states.

A.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 RL methods in this paper. The policies of the other vehicles in the

simulation environment are fixed beforehand. In the overtaking environment, the policy used for the slow leading vehicle (blue in Figure 1) yields the same, fixed actions for every state, keeping the vehicle within the initial lane at a constant velocity. In the driving environment, a mixture of vehicles equipped with a custom rule-based policy and a policy implementing the 'Intelligent Driver Model' (IDM) [22] and 'Minimizing Overall Braking Induced by Lane change' (MOBIL) [10] is used. Both policies try to mimick rudimentary human driving behaviour, although being fully deterministic. Safety of the chosen actions was guaranteed through an extra safety check, similar to what is done by Nageshrao et al. [13]. Unsafe actions are mapped to the nearest safe actions, before being passed to the lower level controllers, avoiding most collisions.

A.4 Reward

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

$$r = w_F r_F + w_V r_V + w_C r_C + w_R r_R + w_B r_B + w_S r_S.$$

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 correctly aligned within its current lane – 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, for some experiments a penalty for non-smooth policies is given in the reward through the r_S component.

The final reward is rescaled by the sum of all composing weights, such that it always lies in the interval $[-1\,;0]$.

B TD3 hyperparameters for the gym environments

The table below shows the used hyperparameters for the TD3 algorithm (and its smoothed variants) on the 10 used OpenAI gym environments used in the experiments section. Most of these values correspond to the tuned hyperparameters of the Stable-Baselines3 repository[7], the differences are highlighted in bold.

[7] https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/hyperparams/td3.yml

Table 3. Overview of the used hyperparameters for each environment. The shown hyperparameters are: maximum timesteps per episode k_M, total training timesteps $k_M \cdot T_M$, distribution of the exploration noise $\epsilon \sim E$, discount factor γ, replay buffer size $|\mathcal{B}|$. The exploration noise generators are: the normal distribution $N(\mu, \sigma)$ with mean μ and standard deviation σ, the Ornstein-Uhlenbeck process $O(\mu, \sigma, \theta)$ with mean μ, standard deviation σ and damping θ.

Environment	$\boldsymbol{k_M}$	$\boldsymbol{k_M T_M}$	$\boldsymbol{E}$	$\boldsymbol{\gamma}$	$\boldsymbol{\lvert\mathcal{B}\rvert}$
Ant (AntBulletEnv-v0)	1000	$1 \cdot 10^6$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
Bipedal (BipedalWalker-v3)	1600	$1 \cdot 10^6$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
Hopper (HopperBulletEnv-v0)	1000	$1 \cdot 10^6$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
IDP (InvertedDoublePendulumBulletEnv-v0)	1000	$1 \cdot 10^6$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
IPS (InvertedPendulumSwingupBulletEnv-v0)	1000	$\mathbf{5 \cdot 10^5}$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
Lunar (LunarLanderContinuous-v2)	1000	$\mathbf{1 \cdot 10^6}$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
Minitaur (MinitaurBulletEnv-v0)	1000	$1 \cdot 10^6$	$N(0, 0.1)$	0.99	$1 \cdot 10^6$
MCC (MountainCarContinuous-v0)	999	$\mathbf{5 \cdot 10^5}$	$O(0, 0.5, 0.15)$	0.99	$1 \cdot 10^6$
Pendulum (Pendulum-v0)	200	$\mathbf{1 \cdot 10^5}$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$
Walker (Walker2DBulletEnv-v0)	1000	$1 \cdot 10^6$	$N(0, 0.1)$	0.98	$2 \cdot 10^5$

Table 4. Overview of the used hyperparameters, common across all used environments.

Common hyperparameters		
Learning rate (actor + critic)	η	$1 \cdot 10^{-3}$
Warmup timesteps		10000
Batch size	B	100
Policy update delay	d	2
Target policy noise distribution		$N(0, 0.2)$
Target policy noise clipping		$[-0.5\,;0.5]$
Polyak averaging constant	τ	$5 \cdot 10^{-3}$
Network architecture – hidden dimensions (actor + critic)		400×300