2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Benchmarking Smoothness and Reducing High-Frequency Oscillations in Continuous Control Policies

Guilherme Christmann*, Ying-Sheng Luo*, Hanjaya Mandala*, and Wei-Chao Chen
Inventec Corporation, Taipei, Taiwan
{*guilherme.christmann, luo.ying-sheng, hsu.hanjaya, chen.wei-chao*}*@inventec.com*

***Abstract*— Reinforcement learning (RL) policies are prone to high-frequency oscillations, especially undesirable when deploying to hardware in the real-world. In this paper, we identify, categorize, and compare methods from the literature that aim to mitigate high-frequency oscillations in deep RL. We define two broad classes: loss regularization and architectural methods. At their core, these methods incentivize learning a smooth mapping, such that nearby states in the input space produce nearby actions in the output space. We present benchmarks in terms of policy performance and control smoothness on traditional RL environments from the Gymnasium and a complex manipulation task, as well as three robotics locomotion tasks that include deployment and evaluation with real-world hardware. Finally, we also propose hybrid methods that combine elements from both loss regularization and architectural methods. We find that the best-performing hybrid outperforms other methods, and improves control smoothness by 26.8% over the baseline, with a worst-case performance degradation of just 2.8%.**

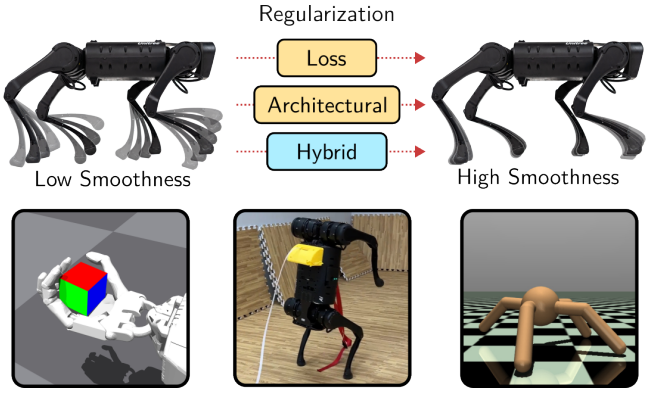

Fig. 1: We investigate the use of different classes of regularization to produce smooth control policies in several simulation and real-world environments.

I. Introduction

Reinforcement learning (RL) policies are prone to high-frequency oscillations. When no limitations or constraints are imposed in either the learning or in the environment, RL agents commonly develop exploitative behavior that maximizes reward to the detriment of everything else. Chasing high task performance (reward) is the goal of learning, but there are scenarios where additional factors must be considered. For example, when deploying a policy to hardware in the real-world high-frequency oscillations are especially undesirable as they can cause damage to the actuators and other hardware.

A straightforward way to mitigate the issue is to include penalization terms as part of the reward function. However, the learning algorithm's tendency to exploit the reward function can lead to policies where subpar performance is preferred in favor of smoothness. Furthermore, reward function design is a complex matter, and can be difficult to express for many tasks in the first place [1], [2], [3]. Adding additional penalization terms for high-frequency oscillations essentially modifies the original learning objective, and can be difficult to tune. If the penalization weight is too large, the policy might prefer to not do much at all to avoid large negative rewards. On the other hand, if the weight is too small it might choose to ignore it and still generate high-frequency oscillations. An ideal method should allow us to maintain the originally designed reward function and remove the need to add new elements of complexity.

Another approach to reducing high-frequency oscillations is to filter the actions outputted from the policy, for example with a low-pass filter. In terms of the classic agent-environment diagram in RL [4], this type of approach can be construed as adding a constraint to the environment rather than to the agent (or policy) itself. In fact, filtering the actions can lead to even larger oscillations in the raw outputs of the policy. That being said, filtering is effective and quite common in practice, especially in robotics applications [5]. A major drawback of using a traditional filter, e.g. a low-pass filter, is that it has memory. This means that if the observation space does not include past actions and past observations the policy will not be able to learn an effective model, as it violates the assumption of a Markov Decision Process [6]. Although this can be solved by keeping a history buffer over multiple steps it requires larger models in terms of parameters and complexity [7].

In this work, we categorize, adapt, and compare methods that aim to mitigate the problem of high-frequency oscillations in deep RL (Figure 1). We focus on methods that do not rely on explicit reward penalization terms, or environment modifications such as post-processing actions. Rather, we identify two classes of methods in the literature: loss regularization and architectural methods. At their core, these methods incentivize or impose constraints such that the

*These authors contributed equally, listed alphabetically by last name.

policy learns to produce smooth mappings between input and output space, without modifying the reward function design. A mapping can be considered smooth when states that are nearby in the input space produce nearby actions in the output space [8]. A common mechanism employed by multiple existing works is to constrain the upper bound of the Lipschitz constant of the policy network [9], either globally [10] or in a local manner [11], [12]. However, we wish to note that in this work we are not concerned with the actual smoothness of the mapping itself, but rather on the observable smoothness in the form of action oscillations during test time.

We benchmark several methods in terms of policy performance and action smoothness. For traditional RL methods we used Gymnasium [13] and for complex manipulation and locomotion we used Isaac Gym [14]. Additionally, for the three locomotion tasks we include evaluation and deployment of every method in the real-world. Our paper also proposes novel hybrid methods that outperform the existing approaches in terms of smoothness and task performance trade-offs. Our contributions can be summarized as follows:

- Categorizing existing methods in two broad classes: loss regularization methods and architectural methods;
- Benchmarking and directly comparing action smoothing methods in classical RL simulation as well as application-focused and complex deployment scenarios in the real-world;
- Proposing novel hybrid methods that combine elements from other existing methods and outperform existing ones.

II. Related Works and Methods Categorization

Benchmarking in RL. Reinforcement learning is a diverse field with diverse tasks and algorithms. With such a large array of possibilities, it is common for practitioners to look for benchmarks to aid in algorithm selection. [15] presented a benchmark for continuous control policies in several classic tasks as well as more complex tasks such as humanoid locomotion in 3D simulation. Other works have performed benchmarks that focus on domains such as meta reinforcement learning [16], manipulation tasks [17], and real-world deployment [18], [19]. In the context of smooth policies, past works have presented brief comparisons as a way to validate their method [12], [11], [7]. However, a comprehensive study that considers a holistic approach does not currently exist.

In our work, we aim to fill this gap in the literature with a comprehensive comparison among methods that learn smooth policies without modifying the original reward function. We identify two major method classes: loss regularization and architectural methods. In the remainder of this section, we describe the general form and characteristics of each class followed by specific methods of that class. We also describe novel hybrid methods that combine elements from both loss regularization and architectural approaches.

A. Loss Regularization Methods

Loss regularization methods aim to reduce the action oscillation frequency by adding regularization components to the standard RL loss objective, rather than directly in the reward function. They have the general form of

$$\mathcal{L} = \mathcal{L}_{\text{RL}} + \mathcal{L}_{Reg} \tag{1}$$

where $\mathcal{L}_{\text{RL}}$ is a policy gradient loss such as PPO [20], TRPO [21], and similar methods; and $\mathcal{L}_{Reg}$ is the regularization loss. We investigate two recent methods from the literature that fit the definition of the loss regularization class.

CAPS [7]. Uses two regularization components. The first is a temporal component $\mathcal{L}_T$, which minimizes the distance between the actions of two consecutive states s_t and s_{t+1}. The second is a spatial component $\mathcal{L}_S$ that minimizes the difference between the state s_t and a state $\bar{s}_t$ sampled from a normal distribution in the neighborhood of s_t. This takes the form of

$$\begin{aligned} \mathcal{L}_T &= D(\pi_\theta(s_t), \pi_\theta(s_{t+1})) \\ \mathcal{L}_S &= D(\pi_\theta(s_t), \pi_\theta(\bar{s}_t)), \quad \text{where } \bar{s}_t \sim \mathcal{N}(s_t, \sigma) \\ \mathcal{L}_{\text{CAPS}} &= \lambda_T \mathcal{L}_T + \lambda_S \mathcal{L}_S \end{aligned} \tag{2}$$

where π_θ is the actor network, $D(\cdot)$ is a distance function, and λ_T, λ_S, and σ are hyperparameters to be tuned. This method is similar to the one proposed by [8], with the main distinction that CAPS uses the L2 distance between sampled actions, while [8] employed KL divergence on the output distributions.

For scenarios in our work that overlap with the original we use the same hyperparameters from the original paper [7]. For the new locomotion and manipulation environments only present in our work, we performed a short hyperparameter search and chose the best values.

L2C2 [11]. Uses two regularization components with a similar mechanism to the spatial component from CAPS. Distinctively, the regularization is employed both to the outputs of the actor-network π_θ and the value network V_θ. Additionally, the sampling distance is bounded relative to the distance of two consecutive states s_t and s_{t+1}, rather than a predefined hyperparameter as in CAPS. The L2C2 regularization is computed in the following way

$$\begin{aligned} \bar{s}_t &= s_t + (s_{t+1} - s_t) \cdot u, \quad \text{where } u \sim \mathcal{U}(.) \\ \mathcal{L}_{s,\pi} &= D(\pi_\theta(s_t), \pi_\theta(\bar{s}_t)) \\ \mathcal{L}_{s,V} &= D(V_\theta(s_t), V_\theta(\bar{s}_t)) \\ \mathcal{L}_{\text{L2C2}} &= \lambda_\pi \mathcal{L}_{s,\pi} + \lambda_V \mathcal{L}_{s,V} \end{aligned} \tag{3}$$

where $\mathcal{U}$ is a uniform distribution, D is a distance metric, π_θ and V_θ are the actor and value network, and λ_π and λ_V are weights for each regularization component. For brevity, the uniform sampling details and its hyperparameters are omitted here. We invite the reader to read the original work from [11] for an in-depth discussion of the state sampling and definition of the hyperparameters.

L2C2 and **CAPS** are similar, with the main difference being the sampling method. It could be argued that the temporal element of **CAPS** is redundant since a state that is sampled nearby and two consecutive states should produce more or less the same regularization signal. As such, **L2C2** drops the temporal element in favor of optimizing both the actor and the value network outputs with a spatial regularization.

B. Architectural Methods

Architectural methods aim to reduce the oscillation frequency of the actions by modifying the learning components of the network. In the case of the Lipschitz based methods [10], [12] they also add an element to the loss function. However, the objective function is used to constrain the upper bound of the Lipschitz value of the network, rather than directly optimizing state-action differences as in the loss regularization category.

Spectral Normalization – Local SN [22]. Spectral normalization is most commonly used to stabilize the training of Generative Adversarial Networks [23]. It consists of a rescaling operation applied to the weights of a layer by its spectral norm $\sigma(\boldsymbol{W})$. The normalized weights are given by $\boldsymbol{W}_{SN} = \delta \cdot \frac{\boldsymbol{W}}{\sigma(\boldsymbol{W})}$. In the context of reinforcement learning, past works have proposed global and local variants of the spectral normalization [22]. The difference between the global and local variants is that spectral normalization is applied to every layer in the global version, and only to the output layer in the local version. In our work, we investigate the local variant **Local SN** due to its significantly better performance reported by the original authors [22].

We implement this method using the spectral normalization existent in *PyTorch*. Our implementation is equivalent to the original description in [22] with a $\delta = 1.0$. This method does not have any other hyperparameters.

Liu-Lipschitz [10]. This approach was originally used to learn a smooth mapping for neural distance field networks, such that interpolation and extrapolation of shapes is possible. The method constrains the Lipschitz upper bound of the network, as a learnable parameter c_i per layer. The weights of each network layer are normalized with regards to c_i and the layer's outputs are computed as such

$$\begin{aligned} y &= \sigma(\hat{W}_i \cdot x + b_i) \\ \hat{W}_i &= normalization(W_i, softplus(c_i)) \end{aligned} \tag{4}$$

where $\hat{W}_i$ are the normalized weights and $\sigma(.)$ is an activation function. For brevity, we omit the implementation details of the normalization procedure and invite the reader to verify the original work [10]. This method also includes a loss function element that minimizes the values of c_i and has the form

$$\mathcal{L}_c = \lambda \prod_i^N softplus(c_i) \tag{5}$$

where λ is a tunable hyperparameter, and N is the number of layers in the network, with a single c_i per layer.

LipsNet [12]. The most recent out of all the methods investigated. It proposes a novel network module called **LipsNet** that can be used as plug and play replacement for a traditional feedforward layer. Specifically, we investigate the best-performing variant **LipsNet-L**, whose output is computed as such

$$y = K(x) \cdot \frac{f(x)}{||\nabla f(x)|| + \epsilon}, \tag{6}$$

where $f(x)$ is a conventional feedforward layer and $||\nabla f(x)||$ is the 2-norm of the Jacobian matrix relative to the input x, $K(x)$ is the Lipschitz value modeled by a feedforward network K conditioned on the input x, and ϵ is a small positive value to avoid division by zero.

The authors of **LipsNet** provided an open-source implementation of their method. However, we noted a few differences from the original description in their work. Specifically, their paper [12] states that the activation of the $K(x)$ module is a softplus function, but in the open-source code a linear activation was used. In our implementation, we used a softplus activation as described in the original paper[1]. Additionally, we opted to not use a $tanh$ squashing function in the outputs of the network and instead use a linear activation, the same as every other method we experiment with in this work.

III. Method and Experimental Setup

All experiments are run using PPO with a focus on continuous observations and continuous action spaces. We benchmark traditional RL environments using Gymnasium [13], and robotics application scenarios with Isaac Gym [14] and sim2real deployment to real hardware [24], [25]. The base PPO implementations used are *Stable Baselines* [26] for the Gymnasium environments and the *RL Games* [27] implementation for Isaac Gym. We extended these implementations with support for all the methods outlined in Section II. All of our implementations are written using *PyTorch*.

For every environment and every method, we trained policies from scratch using 9 different random seeds. Where applicable, we utilized the same hyperparameters for the same environments presented in the original works. In other cases, we tuned the parameters for better performance. The complete hyperparameters used in our investigation are presented in Table I.

A. Hybrid Methods

We investigate the effectiveness of hybrid methods that combine elements from architectural as well as loss regularization approaches. Specifically, we focus on the combination of a **LipsNet** style network with additional regularization components in the style of **L2C2** and **CAPS**. We exclude

[1]We have contacted the authors of **LipsNet** and verified that our implementation as described in this work indeed reflects the original one used in their experiments. The open-source implementation has since been corrected to use a softplus activation.

Method	Parameter	Gymnasium	ShadowHand	Motion Imitation	Velocity	Handstand
CAPS	σ	0.1	0.2	0.2	0.2	0.2
	λ_T	0.1	0.01	0.01	0.01	0.01
	λ_S	0.5	0.05	0.05	0.05	0.05
L2C2	σ	1.0	1.0	1.0	1.0	1.0
	$\underline{\lambda}$	0.0	0.01	0.01	0.01	0.01
	$\overline{\lambda}$	1.0	1.0	1.0	1.0	1.0
	β	0.1	0.1	0.1	0.1	0.1
LipsNet	Weight λ	0.1	0.00001	0.001	0.001	0.0001
	ϵ	0.0001	0.0001	0.0001	0.0001	0.0001
	Initial Lipschitz constant K_{init}	1.0	1.0	1.0	1.0	1.0
	Hidden layers in $f(x)$	[64, 64]	[512, 256]	[512, 256]	[512, 256]	[512, 256, 128]
	Activation in $f(x)$	ELU	ELU	ELU	ELU	ELU
	Hidden layers in $K(x)$	[32]	[32]	[32]	[32]	[32]
	Activation in $K(x)$	Tanh	Tanh	Tanh	Tanh	Tanh
Liu-Lipschitz	Weight λ	1×10^{-6}	1×10^{-7}	1×10^{-6}	1×10^{-5}	1×10^{-6}
	Initial Lipschitz constant	10.0	10.0	10.0	1.0	10.0

TABLE I: Hyperparameters used during training for every method.

Local SN due to inferior performance and inferior training stability and exclude **Liu-Lipschitz** due to the method similarity with **LipsNet** but inferior performance. We propose, experiment, and analyze two novel hybrid methods: **LipsNet + CAPS**, and **LipsNet + L2C2**.

B. Metrics

Cumulative Return. The cumulative sum of the reward at every step throughout a whole episode $C = \sum_{t=0}^{N} R_t$. It provides a measure of the task performance of the policy. This metric is environment dependent and is used primarily to analyze the trade-off between smoothness and performance.

Smoothness. We adopt the same smoothness metric as [7], computed from the frequency spectrum of a Fast Fourier Transform (FFT). The smoothness measure Sm computes a normalized weighted mean frequency and has the form

$$Sm = \frac{2}{n\ f_s} \sum_{i=1}^{n} M_i f_i, \tag{7}$$

where n is the number of frequency bands, f_s the sampling frequency, and, M_i and f_i are the amplitude and frequency of band i, respectively. Higher values indicate the presence of high-frequency components of large magnitude, and lower values indicate a smoother control signal. In the same manner as the cumulative return, a good smoothness value differs from environment to environment but is independent of the policy control frequency.

C. Evaluation Scenarios

Gymnasium Baselines. Gymnasium [13] provides standard and classical RL environments for easy and diverse comparisons across different algorithms. We use it to evaluate 4 classic continuous control environments: Pendulum-v1, Reacher-v4, LunarLander-v2 (Continuous version), and Ant-v4. Because Pendulum-v1 is a simpler environment we train the policies for just 150k timesteps, while the remaining environments train for a total of 400k timesteps. For evaluation, the metrics are computed from 1000 independent episodes for each training seed and averaged.

Robotics Applications. With Isaac Gym [14] we train policies to execute three locomotion tasks with a quadruped robot and a manipulation task with a highly dexterous hand. The manipulation task *ShadowHand* is one of the standard tasks bundled with Isaac Gym and consists of manipulating a cube to match a target orientation [28]. The other three tasks are implemented by us with additional deployment on real-world hardware. *Motion Imitation*, where the agent is rewarded for matching the states of a motion-captured pacing animation [29], [5], [30]; *Velocity* where the agent is rewarded for matching a velocity vector [31]. Locomotion emerges as the result of tracking the velocity command plus additional regularization terms. Note that we do not use an action penalization term in the reward design of this task as was done in past works [31]; and *Handstand*, where the agent is rewarded for standing on its hind legs and maintaining an upright orientation by tracking a target orientation vector. The task includes additional reward regularization terms to minimize joint changes as well as linear and angular velocities. *Motion Imitation* and *Handstand* task are trained for 150M timesteps, and *Velocity* and *ShadowHand* are trained for 300M timesteps. For each training seed, the evaluation metrics are collected and averaged from 10k trajectories for *Motion Imitation* and *ShadowHand* and 50k trajectories for *Velocity* and *Handstand*.

Parameter	Value	Type
Action Noise	0.02	Additive
Rigid Bodies Mass	[0.95, 1.05]	Scaling
Stiffness Gain (PD Controller)	[-10%, +10%]	–
Damping Gain (PD Controller)	[-15%, +15%]	–
Ground Friction	[0.1, 1.5]	–
Sensor Noise - Orientation	0.06	Additive
Sensor Noise - Linear Velocity	0.25	Additive
Sensor Noise - Angular Velocity	0.3	Additive
Sensor Noise - Joint Angles	0.02	Additive
Sensor Noise - Feet Contacts	0.2	Probability

TABLE II: Domain randomization parameters used to train the policies that are deployed to the real-world.

Real-World Experiments. For the three locomotion tasks described above, we conduct real-world experiments with

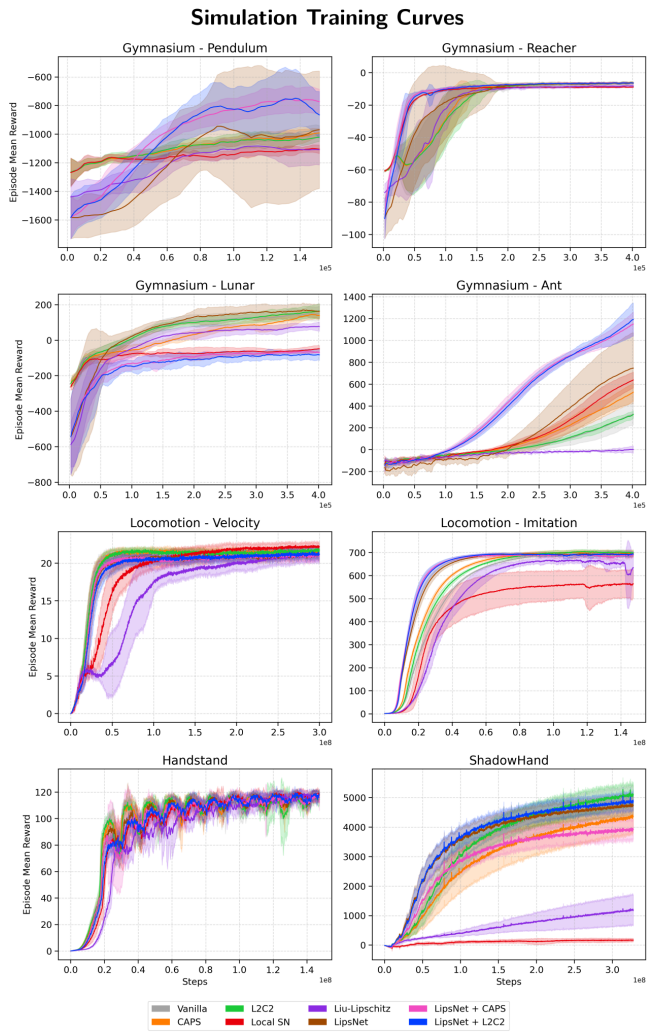

Fig. 2: Reward curves during training for 9 seeds. The hybrid methods **LipsNet + CAPS** and **LipsNet + L2C2** show superior or comparable all environments, except in *Lunar*.

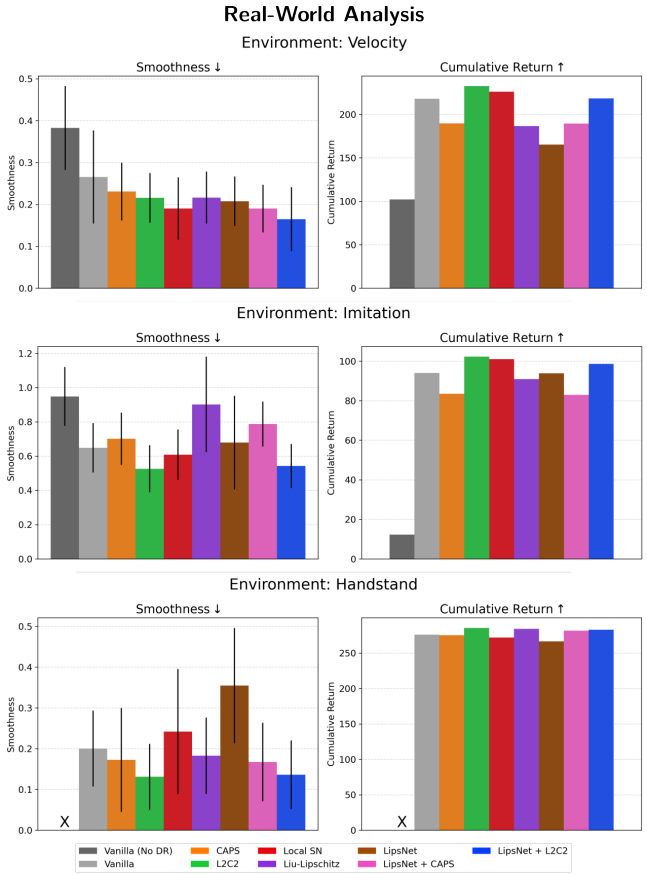

Fig. 3: The methods are evaluated with the real-world robot. Every method achieves similar task performance (measured as cumulative return). The hybrid methods consistently outperformed other methods in regards to smoothness compared to the other methods.

a quadruped robot. The policies are trained with domain randomization (DR) [32] to ensure a successful sim-to-real transfer. By adding noise to elements of the simulation the policy learns to perform reasonably across a larger distribution of states. The simulation parameters randomized during training of the deployment version of the policies are presented in Table II. We investigate the effect of every single method in the real-world, with an additional ablation case where a vanilla policy is trained without DR. This case demonstrates that the use of DR already produces smoother control policies. The evaluation metrics at deployment time in the real-world are computed from 6-second trajectories recorded during the execution of the policies. We compute the smoothness Sm of the whole 6-second trajectory, and the cumulative return is the accumulated reward value from each step throughout the whole trajectory. Note that for the Handstand task, we do not deploy a Vanilla policy without domain randomization, due to the risk of hardware damage from bad policy performance and large oscillations.

IV. Experiments and Results

We benchmarked a total of 7 distinct approaches plus a vanilla baseline, across 8 different scenarios, including classic RL environments and complex robotic simulations with additional deployment in the real-world. The complete results of our simulation benchmark are consolidated in Table III. Additionally, for sample efficiency analysis, the training curves showcasing the episode mean reward over time are depicted in Figure 2.

From the simulation results presented in Table III we can infer that every method improves smoothness compared to the **Vanilla** policy. However, this comes with a performance cost in some cases. The environments *Ant* and *Reacher* had a high-performance variance, with many methods performing significantly worse than the baseline. Other environments provide more consistent results and serve better for a smoothness-performance tradeoff analysis. We can observe that the loss regularization methods **CAPS** and **L2C2** perform similarly in most cases, with a decent improvement in smoothness and a small performance hit overall. On the architectural methods, **Local SN** and **Liu-**

Cumulative Return $\uparrow$

Environment	Pendulum	Ant	Reacher	Lunar
Vanilla	-944 ± 57	833 ± 110	-6.05 ± 0.46	170 ± 49
CAPS – [7]	-940 ± 51	1027 ± 135	$\mathbf{-5.98 \pm 0.21}$	-117 ± 43
L2C2 – [11]	-962 ± 44	791 ± 104	-6.14 ± 0.49	$\mathbf{192 \pm 32}$
Local SN – [22]	-1099 ± 47	1108 ± 174	-8.73 ± 0.31	-126 ± 28
Liu-Lipschitz – [10]	-1056 ± 111	137 ± 210	-7.68 ± 0.86	92 ± 70
LipsNet – [12]	-934 ± 445	959 ± 506	-6.34 ± 0.69	114 ± 71
LipsNet + CAPS – Hybrid	$\mathbf{-737 \pm 182}$	$\mathbf{1683 \pm 228}$	-6.13 ± 0.30	-304 ± 22
LipsNet + L2C2 – Hybrid	-870 ± 172	1684 ± 415	-6.27 ± 0.44	-281 ± 72
Environment	ShadowHand	Imitation	Velocity	Handstand
Vanilla	5025 ± 460	697 ± 19	5.98 ± 0.05	3.37 ± 0.10
CAPS – [7]	4421 ± 576	689 ± 26	$\mathbf{5.99 \pm 0.03}$	3.35 ± 0.06
L2C2 – [11]	$\mathbf{5190 \pm 390}$	$\mathbf{697 \pm 17}$	5.86 ± 0.04	3.41 ± 0.06
Local SN – [22]	166 ± 93	522 ± 132	5.71 ± 0.19	3.37 ± 0.07
Liu-Lipschitz – [10]	1213 ± 538	644 ± 53	5.34 ± 0.12	3.38 ± 0.03
LipsNet – [12]	4784 ± 333	682 ± 27	5.86 ± 0.12	3.40 ± 0.06
LipsNet + CAPS – Hybrid	3923 ± 347	673 ± 27	5.91 ± 0.06	3.40 ± 0.06
LipsNet + L2C2 – Hybrid	4913 ± 305	678 ± 33	5.83 ± 0.13	$\mathbf{3.45 \pm 0.05}$

Smoothness $Sm \downarrow$

Environment	Pendulum	Ant	Reacher $\cdot 10^1$	Lunar $\cdot 10^1$
Vanilla	0.77 ± 0.04	1.94 ± 0.57	0.62 ± 1.08	6.24 ± 7.05
CAPS – [7]	0.73 ± 0.06	1.11 ± 0.40	0.52 ± 0.05	2.20 ± 0.66
L2C2 – [11]	0.73 ± 0.08	1.49 ± 0.43	0.56 ± 0.09	5.43 ± 0.78
Local SN – [22]	0.40 ± 0.07	$\mathbf{0.70 \pm 0.29}$	0.60 ± 0.11	3.90 ± 0.36
Liu-Lipschitz – [10]	0.49 ± 0.11	1.07 ± 0.28	0.47 ± 0.13	6.23 ± 0.76
LipsNet – [12]	0.94 ± 0.42	1.38 ± 0.36	1.11 ± 1.23	5.50 ± 2.96
LipsNet + CAPS – Hybrid	$\mathbf{0.31 \pm 0.08}$	0.75 ± 0.10	$\mathbf{0.35 \pm 0.05}$	$\mathbf{0.59 \pm 0.11}$
LipsNet + L2C2 – Hybrid	0.64 ± 0.28	0.87 ± 0.17	0.35 ± 0.11	0.95 ± 0.21
Environment	ShadowHand	Imitation	Velocity	Handstand
Vanilla	1.82 ± 0.09	0.68 ± 0.16	0.40 ± 0.01	0.64 ± 0.11
CAPS – [7]	1.63 ± 0.15	0.70 ± 0.16	0.40 ± 0.02	0.63 ± 0.07
L2C2 – [11]	1.64 ± 0.07	0.52 ± 0.15	0.52 ± 0.15	0.56 ± 0.05
Local SN – [22]	0.09 ± 0.08	0.63 ± 0.06	0.35 ± 0.19	0.58 ± 0.04
Liu-Lipschitz – [10]	1.28 ± 0.2	0.66 ± 0.10	$\mathbf{0.24 \pm 0.11}$	0.60 ± 0.03
LipsNet – [12]	1.77 ± 0.07	0.65 ± 0.13	0.30 ± 0.10	0.55 ± 0.03
LipsNet + CAPS – Hybrid	$\mathbf{1.58 \pm 0.06}$	0.60 ± 0.12	0.28 ± 0.07	0.56 ± 0.06
LipsNet + L2C2 – Hybrid	$\mathbf{1.57 \pm 0.08}$	$\mathbf{0.52 \pm 0.07}$	$\mathbf{0.26 \pm 0.06}$	$\mathbf{0.46 \pm 0.04}$

TABLE III: Benchmark of task performance and smoothness of different algorithms in the literature. Each method is trained from scratch with 9 different seeds. The table shows the mean and 1 standard deviation of smoothness and return for 9 seeds. Pendulum, Ant, Reacher and Lunar are Gymnasium environments, and Imitation and Velocity are locomotion tasks in Isaac Gym.

Lipschitz generated even smoother policies, at the cost of a large performance deficit. **LipsNet** is inconsistent, sometimes generating smooth policies (see *Velocity* and *Handstand*) and others performed significantly worse than the baseline (see *Ant* and *Reacher*).

Our hybrid methods **LipsNet + CAPS** and **LipsNet + L2C2** outperforms the existing methods in nearly every environment. The *Lunar* environment is the single exception where they are clearly inferior, with better smoothness but low cumulative return. We hypothesize that situations like this might happen due to the policy "getting stuck" too early in optimizing for smoothness, rather than task performance. More extensive hyperparameters search and better scheduling of learning rate and loss weights could yield better outcomes. Excluding *Ant* and *Lunar* due to high variance across methods, we can observe that **LipsNet + CAPS** produces 28.4% smoother control compared to the **Vanilla** baseline, while **LipsNet + L2C2** is at a close second with a 26.8% average improvement. In terms of performance impact, **LipsNet + CAPS** had a worst case performance degradation of 21.9% in the *ShadowHand* environment. On the other hand, **LipsNet + L2C2** reproduced the performance of the unregularized **Vanilla** more consistently, with a worst case degradation of only 2.8% in *Imitation*.

For the three robotics locomotion tasks *Imitation*, *Velocity*, and *Handstand* we perform deployment and analysis with a real-world quadruped robot. The complete results of the real-world deployment are presented in Figure 3, which includes the **Vanilla** baseline and the 7 methods investigated in this work, plus an additional ablation of a **Vanilla** policy trained without domain randomization. Domain randomization is commonly used as a way to improve real-world deployment performance. In this work, we also note that DR significantly improves the smoothness of the control, on top of the expected performance improvement. Analyzing the bar plots we can observe that many of the methods impact measured in simulation are diminished in the real-world with the inclusion of DR. In *Velocity* we can observe smoothness improves with every method compared to the **Vanilla** baseline. However, the other two scenarios *Imitation* and *Handstand* have more variance, with several methods ending up less smooth than the baseline with DR. Still, **L2C2** and the hybrid **LipsNet**

+ L2C2 are the clear superior methods in the real-world experiment, outperforming every other approach. Both methods significantly improved smoothness over **Vanilla** while maintaining comparable task performance.

During our real-world experiments we also observed an emergent behavior when the agent suffers from large disturbances. For example, when the robot is lifted on the air or flipped over, **Vanilla** policies tend to generate high-frequency oscillations in sudden bursts, which could risk hardware damage. On the other hand, the regularized policy **LipsNet + L2C2** elegantly stops execution until the agent is reset to the ground in an upright position. We invite the reader to check the video in the supplementary material for demonstrations of this behavior.

V. Conclusion

In this work we present a benchmark of methods that can reduce the frequency of control oscillations in policies learned with reinforcement learning. We identify 7 methods from the literature and classify them according to their mechanism. We propose two broad method classes: loss regularization and architectural methods. Loss regularization methods rely purely on adding regularization elements to the standard policy gradient loss. On the other hand, architectural methods introduce modifications of network elements such as weight normalization and specialized modules that can replace traditional feedforward layers. Additionally, we also introduce and investigate two novel hybrid methods that combine properties from both method classes.

Our benchmark includes 4 traditional RL environments and 4 complex robotics tasks involving manipulation and locomotion. We analyze every method regarding task performance as well as the smoothness of output actions. In general, the investigated methods perform better than the unregularized **Vanilla** baseline in every scenario, with a few exceptions. We identify **LipsNet + L2C2** as the best-performing method in simulation, with a smoothness improvement of 26.8% over the baseline, and a worst case task performance degradation of just 2.8%. For the three robotics tasks that involve locomotion, we deploy and perform analysis in the real-world. Overall, **L2C2** and hybrid method **LipsNet + L2C2** showed the best trade-off between smoothness and task performance in the real-world. As such, we recommend that practitioners concerned with oscillations train policies with one of these approaches.

In the future, we wish to investigate tasks with an emphasis on more diverse robotics applications. Reducing high-frequency oscillations is essential for successful sim-to-real transfer and to prevent hardware damage. As such, the community can benefit from a clearer set of guidelines on how to train policies for tasks such as locomotion, pick and place, contact rich manipulation, etc.

References

[1] A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine, "Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 281–15 295, 2022.

[2] A. Laud and G. DeJong, "The influence of reward on the speed of reinforcement learning: An analysis of shaping," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 440–447.

[3] J. Eschmann, "Reward function design in reinforcement learning," *Reinforcement learning algorithms: Analysis and Applications*, pp. 25–33, 2021.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[6] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning: State-of-the-art*. Springer, 2012, pp. 3–42.

[7] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, "Regularizing action policies for smooth control with reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 1810–1816.

[8] Q. Shen, Y. Li, H. Jiang, Z. Wang, and T. Zhao, "Deep reinforcement learning with robust and smooth policy," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8707–8718.

[9] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] H.-T. D. Liu, F. Williams, A. Jacobson, S. Fidler, and O. Litany, "Learning smooth neural functions via lipschitz regularization," in *ACM SIGGRAPH 2022 Conference Proceedings*, 2022, pp. 1–13.

[11] T. Kobayashi, "L2c2: Locally lipschitz continuous constraint towards stable and smooth reinforcement learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4032–4039.

[12] X. Song, J. Duan, W. Wang, S. E. Li, C. Chen, B. Cheng, B. Zhang, J. Wei, and X. S. Wang, "Lipsnet: a smooth and robust neural network with adaptive lipschitz constant for high accuracy optimal control," in *International Conference on Machine Learning*. PMLR, 2023, pp. 32 253–32 272.

[13] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[14] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[15] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*. PMLR, 2016, pp. 1329–1338.

[16] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*. PMLR, 2020, pp. 1094–1100.

[17] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Conference on Robot Learning*. PMLR, 2018, pp. 767–782.

[18] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Conference on robot learning*. PMLR, 2018, pp. 561–591.

[19] N. Gürtler, S. Blaes, P. Kolev, F. Widmaier, M. Wüthrich, S. Bauer, B. Schölkopf, and G. Martius, "Benchmarking offline reinforcement learning on real-robot hardware," *arXiv preprint arXiv:2307.15690*, 2023.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[22] R. Takase, N. Yoshikawa, T. Mariyama, and T. Tsuchiya, "Stability-

certified reinforcement learning control via spectral normalization," *Machine Learning with Applications*, vol. 10, p. 100409, 2022.

[23] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[24] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[25] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[26] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[27] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning," https://github.com/Denys88/rl_games, May 2021.

[28] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[29] X. B. Peng, P. Abbeel, S. Levine, and M. Van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions On Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.

[30] G. Christmann, Y.-S. Luo, J. H. Soeseno, and W.-C. Chen, "Expanding versatility of agile locomotion through policy transitions using latent state representation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5134–5140.

[31] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," 2021.

[32] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.