Locally Asymptotically Stable Deep Actor-Critic Algorithms For Quadratic Cost Setpoint Optimal Control Problems

ABSTRACT

These recent years have seen the surge of deep reinforcement learning (DRL) as a natural tool to solve complex nonlinear control problems. A notable example is to find a control policy that robustly steers the state to the desired setpoint—*i.e.,* finding a stabilizing control policy. Though few DRL algorithms eventually converge to control policies equipped with stability guarantees, none of them can maintain these properties throughout the training phase, thus restricting their applicability. In this paper, however, we build upon a recent theory enhancing DRL with local dynamics information to ensure (local) stability on the fly. In contrast, we circumvent the need for local dynamic knowledge by learning it from data. We then introduce an algorithmic scheme, compatible with many classical DRL methods, which preserves stability properties throughout the training phase despite learned quantities. Experiments conducted over three benchmarks and four classical DRL algorithms—*i.e.,* PPO, SAC, TD3, and DDPG—support these claims.

KEYWORDS

Stability, Deep Reinforcement Learning, Robot control

1 INTRODUCTION

Setpoint optimal control tasks aim at controlling a dynamical system such that it optimally reaches a constant equilibrium point [32]. A *setpoint*, or equilibrium point, is a state-action pair such that the state does not change upon the application of the action [21, Chapter 4]. These problems are widespread in the industry. Asking a humanoid robot to stand up and remain balanced, moving an industrial robotic arm in a predefined configuration, or requiring a drone to reach a target location, are notable examples. In all these applications, we usually desire to control the system so that to perfectly reach the setpoint starting from any initial condition. Moreover, we want such a behavior to be robust to environment perturbations. To do so, two properties are typically required: *global convergence* and *local asymptotic stability* (LAS). The global convergence properties steer any initial state to a neighborhood of the setpoint. Asymptotic stability (AS) ensures the distance to the setpoint robustly decreases to zero as time evolves. This property is referred to as LAS whenever AS holds only in the neighborhood of the setpoint [5].

Control theory provides standard solutions, *i.e.,* stabilizing policies, to setpoint optimal control tasks. These policies come with stability guarantees for the closed-loop system. The reader can find multiple formulations of stability in the literature depending on the problem to be studied [1, 21, 29]. Yet, in setpoint control problem the most employed definition is asymptotic stability in the sense of Lyapunov [21, Chapter 4]. This stability property ensures a reliable behavior in the presence of slight deviations from the nominal environment. It is necessary for real-world scenarios to withstand parametric uncertainties, noise, or external disturbances and ensure exact convergence to the desired setpoint. In other words, stability guarantees robustness with respect to unseen data, therefore it ensures *generalization*. Stabilizing policies are typically issued from control theory and rely on the knowledge of the dynamics of the system. However, when dealing with complex nonlinear systems, finding a good description of the dynamics is non-trivial, let alone ensuring stability over the entire state space. One possibility is to rely on optimization based methods [14, 15]. A fall-back approach in the control-theoretic community provides LAS guarantees based on a simplified model around the setpoint. Doing so significantly impacts the behavior of the closed-loop system once far from it. Policies with LAS apply in the proximity of the setpoint, known as the *domain of attraction*. Yet, they lack the global convergence property. Thus, outside the domain of attraction, the closed-loop system can be unstable, which typically means unreliable and potentially dangerous.

The last decade experienced the surge of DRL algorithms for solving nonlinear and unstructured control problems [20, 23, 24, 27, 28]. An intrinsic property of such algorithms is value generalization over instances of the same problem. If reward signals encourage convergence to the setpoint, then value generalization may provide global convergence (in the aforementioned sense). Indeed, algorithms of this family guarantee value to reach a local optimum, which may get closer to a global optimum with more data. In real-world setpoint control tasks, however, global convergence is often not sufficient as we also require LAS. We can identify three main classes of approaches trying to close the gap between DRL and real-world applications: Safe Reinforcement Learning techniques, methods achieving LAS after training and algorithms guaranteeing LAS even during the training phase. Safe Reinforcement Learning approaches requires the system not to visit some unsafe state-action regions [16]. However, stability demands for asymptotic convergence to the setpoint. Thus, stability differs from safety, as the latter does not guarantee such an asymptotic behavior. A recent line of research incorporates solutions from the control-theoretic community into DRL to eventually converge to stabilizing controllers [3, 9–11, 17, 19, 33, 34]. Berkenkamp et al. [3] relied on a learned Gaussian model to certify Lyapunov stability, providing we know a Lyapunov function. Han et al. [19] designed a critic such that they learn a Lyapunov function. Most existing approaches asymptotically learn stability but provide no guarantee throughout the training process. Zhang and Capel [33], Zoboli et al. [34], however, provided theoretical frameworks to enhance many DRL algorithms with LAS, including at the training phase. Such a property is critical when learning from interactions with (real) cyber-physical systems. Unfortunately, these approaches assume access to local dynamics information around a given setpoint. Then, control theory provides them with tools to compute a stabilizing policy that applies only locally, *i.e.,* the local stabilizing policy, which comes along with a domain of attraction. Finally, they learn a neural policy, which operates outside the domain of attraction. Yet, neither Zoboli et al. [34] nor Zhang and Capel [33]

succeeded in providing an end-to-end algorithm along with practical experimental validations.

In this paper, we show that aforementioned approaches guaranteeing LAS during the training phase are effective, even when we have no access to the local dynamics information. As the global policy is guaranteed to be locally stabilizing, it can withstand bounded perturbations [35]. Then, to extend such approaches, we rely on the robustness provided by the local stabilizing policy to compensate the model errors and we learn the required local quantities offline. We identify three required quantities including the setpoint, the local model, and the domain of attraction. We first estimate the setpoint, which gives the action, providing we know the goal state. Next, we learn the unknown local model around the estimated setpoint. Finally, we estimate the domain of attraction of the local stabilizing policy from the learned model. Then, we combine these estimated quantities and design an end-to-end local asymptotically stable (LAS) DRL algorithm. This algorithm is guaranteed to be LAS during and after the training of parameters. We conduct experiments against standard deep actor-critic algorithms, *e.g.,* PPO [27], DDPG [23], SAC [18], and TD3 [13], over three benchmarks building upon standard domains from DRL literature. We are interested in showing the proposed approach still provides robust policies equipped with local stability guarantees. Results confirm the method succeeds at embedding LAS in neural agent despite estimated quantities. Moreover, they emphasize the importance of enforcing stability properties in learned controllers to gain generalization and robustness.

2 BACKGROUND

In this section, we formalize the setpoint optimal control problem and the theoretical background necessary to embed LAS properties in DRL algorithms.

2.1 Setpoint Optimal Control Problems

A discrete-time setpoint optimal control problem under a discounted quadratic cost is given by a tuple $M \doteq (\mathcal{X}, \mathcal{U}, R_\gamma, Q_\gamma, f, \gamma, x^\star, u^\star)$ where $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{U} \subseteq \mathbb{R}^m$ is the action space, $Q_\gamma = Q_\gamma^\top \geq 0$ and $R_\gamma = R_\gamma^\top > 0$ are symmetric weight matrices of appropriate dimensions, $\gamma \in (0, 1)$ is a discount factor, $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the unknown nonlinear dynamics, and $(x^\star, u^\star)$ is the setpoint with given goal state $x^\star$ and unknown action $u^\star$.

Solving M aims at finding a policy $\pi^\star : \mathcal{X} \to \mathcal{U}$, which reaches $(x^\star, u^\star)$ optimally with respect to the cumulative sum of quadratic costs. Cost function $\mathrm{J}^\pi : \mathcal{X} \to \mathbb{R}$ denotes the cumulative sum of quadratic costs under policy π for any arbitrary state $x \in \mathcal{X}$, *i.e.,*

$$\mathrm{J}^\pi(x) \doteq \sum_{k=0}^{\infty} \gamma^k \left(\tilde{x}_k^\top Q_\gamma \tilde{x}_k + \tilde{\pi}(x_k)^\top R_\gamma \tilde{\pi}(x_k) \right), \tag{1}$$

where $\tilde{x} \triangleq x - x^\star$, $\tilde{u} \triangleq u - u^\star$, $\tilde{\pi}(x) \triangleq \pi(x) - u^\star$, and x_k is the $k-$steps ahead state starting from x. The optimal cost function $\mathrm{J}^\star : \mathcal{X} \to \mathbb{R}$ provides the minimum cumulative sum of quadratic costs for any arbitrary state $x \in \mathcal{X}$, *i.e.,* $\mathrm{J}^\star(x) \doteq \min_\pi \mathrm{J}^\pi(x)$. Similarly, optimal action-cost function $\mathbf{Q}^\star : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ provides the minimum cumulative sum of quadratic costs for any arbitrary state-action pair $(x, u) \in \mathcal{X} \times \mathcal{U}$, *i.e.,*

$$\mathbf{Q}^\star(x, u) \doteq \tilde{x}^\top Q_\gamma \tilde{x} + \tilde{u}^\top R_\gamma \tilde{u} + \gamma \mathrm{J}^\star(f(x, u)). \tag{2}$$

We derive an optimal policy $\pi^\star : \mathcal{X} \to \mathcal{U}$ greedily based upon optimal action-cost function $\mathbf{Q}^\star$ for any arbitrary state $x \in \mathcal{X}$ as follows $\pi^\star(x) = \arg\min_u \mathbf{Q}^\star(x, u)$. Intuitively, M is a deterministic and continuous Markov decision process with a discounted quadratic cost function. Such a framework is well suited for describing the control of physical systems.

Under mild conditions on discount factor γ, an optimal policy for M ensures asymptotic stability [25]. Unfortunately, computing an optimal policy for M is non-trivial, which explains why we often rely on approximate ones. However, not all approximate policies guarantee asymptotic stability. As a consequence, we impose constraints on approximate policies to guarantee LAS. A policy is said to provide LAS if it makes the setpoint locally asymptotically stable. For a controlled nonlinear system, a setpoint is said to be locally asymptotically stable if there exist a region around it such that, if the state lies inside such a region, its evolution under the control policy converges to such equilibrium setpoint. Mathematically, a setpoint is said to be locally asymptotically stable if $\forall \varepsilon > 0$, $\exists \delta = \delta(\varepsilon) > 0$ such that $\|x - x^\star\| < \delta$ implies $\lim_{k \to \infty} \|x_k - x^\star\| = 0$ [5]. In the following we assume the discount factor satisfies the conditions of [25]. Then, we assume a stabilizing policy exists, at least locally.

2.2 Locally Asymptotically Stabilizing DNN Policy

This subsection provides an overview of the theoretical foundation necessary to embed LAS guarantees in DRL algorithms [33, 34]. We also highlight the limitations that restrict its applicability.

Previous works exploited knowledge about the local linear dynamics around the desired setpoint to obtain composite policy $\boldsymbol{\pi}^\theta$ that exhibits both LAS and global convergence. This composite (neural) policy is made up of two policies: local stabilizing policy $\boldsymbol{\pi}_{\mathrm{loc}}$ and global one $\boldsymbol{\pi}^\theta_{\mathrm{glo}}$. The local stabilizing policy ensures LAS of the desired setpoint and inside the domain of attraction. The global policy trained through DRL achieves global convergence outside the domain of attraction. Local and global policies combine one another through a smooth blending function, which weights them *wrt* the distance from the setpoint. In particular, composite policy $\boldsymbol{\pi}^\theta$ is the global policy as we get far from the setpoint. It is the local stabilizing policy the closer we get to the setpoint. As a result, composite policy $\boldsymbol{\pi}^\theta$ takes the form, for any arbitrary state $x \in \mathcal{X}$,

$$\begin{aligned} \boldsymbol{\pi}^\theta(x) &= \boldsymbol{\pi}_{\mathrm{loc}}(x) + \boldsymbol{\pi}^\theta_{\mathrm{glo}}(x) \\ \boldsymbol{\pi}_{\mathrm{loc}}(x) &= u^\star + K^\star(x - x^\star) \\ \boldsymbol{\pi}^\theta_{\mathrm{glo}}(x) &= h_1(x) \left(\boldsymbol{\mu}^\theta(x) - \boldsymbol{\pi}_{\mathrm{loc}}(x) \right), \end{aligned} \tag{3}$$

where gain matrix $K^\star$ embeds in the local stabilizing policy $\boldsymbol{\pi}_{\mathrm{loc}}$ the knowledge of the local model. The definition of $K^\star$ can be found in Appendix A. Function $h_1 : \mathcal{X} \to \mathbb{R}$ acts as the blending function. Its presence is fundamental to guarantee the policy shows a locally stable behavior. As a matter of fact, simple addition of a learned component to a locally stabilizing policy does not guarantee the composite policy will be stabilizing. Indeed, the learned portion may overcome the local one, disrupting the stability properties of the composite policy. On the contrary, a proper choice of the h_1 function ensures the local policy dominates the global one once close to the setpoint. This function saturates to 1 far from the setpoint, it locally behaves as a higher order function with respect to $\boldsymbol{\pi}_{\mathrm{loc}}$. It is 0 in the

goal state, and it weights the local and global parts of the control $\boldsymbol{\pi}^\theta(x)$. The component $\boldsymbol{\mu}^\theta$ is a function to be learned, *e.g.* the deep network, it is parametrized by the set of parameters $\theta \in \mathbb{R}^p$ and it is assumed to be locally Lipschitz. From Equation (3) it can be seen that if $h_1(x) = 0$ then the policy is exactly $\boldsymbol{\pi}_{\text{loc}}$, which provides LAS. Conversely, if $h_1(x) = 1$ then $\boldsymbol{\pi}^\theta(x) = \boldsymbol{\mu}^\theta(x)$, which provides global convergence.

Given the policy structure, *i.e.,* Equation (3), and the dynamic model, we can shape a composite action-cost function $\hat{\mathrm{Q}}^\phi_{\boldsymbol{\pi}} \colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ similarly to the composite policy. This choice renders the estimation exact in the setpoint and provides a good approximation in a neighborhood of it. The critic is shaped as

$$\begin{aligned} \hat{\mathrm{Q}}^\phi_{\boldsymbol{\pi}}(x,u) &= \mathrm{Q}_{\text{loc}}(x,u) + \hat{\mathrm{Q}}^\phi_{\text{glo}}(x,u) \\ \mathrm{Q}_{\text{loc}}(x,u) &= (z - z^\star)^\top H (z - z^\star) \\ \hat{\mathrm{Q}}^\phi_{\text{glo}}(x,u) &= h_2(x)(\Omega^\phi(x,u) - \mathrm{Q}_{\text{loc}}(x,u)), \end{aligned} \tag{4}$$

where h_2 plays the same saturation role as h_1 in (3), Ω^ϕ is a parametrized function whose parameter vector $\phi \in \mathbb{R}^q$ is learned by the agent and it is assumed to be locally Lipschitz. Finally, the state-input vectors $z^\top := (x^\top u^\top)$, $z^{\star\top} := (x^{\star\top} u^{\star\top})$ and H is a matrix embedding the knowledge of the action-value function for the linearized system under the optimal local stabilizing policy $\boldsymbol{\pi}_{\text{loc}}$. The definition of H can be found in Appendix A.

Unfortunately, neither [34] or [33] provide a practical end-to-end algorithm. This was prevented by three assumptions. First, they assumed the knowledge of the goal action $u^\star$ generating a setpoint, i.e. if we apply $u^\star$ in $x^\star$ the next state will remain $x^\star$. Second, they assume a linearized model approximating the state evolution around the setpoint $(x^\star, u^\star)$ is known. Finally, it is assumed some knowledge about the size of the domain of attraction of $\boldsymbol{\pi}_{\text{loc}}$. In the following Section, we show that a simple routine can overcome these strong assumptions.

3 ESTIMATING UNKNOWN QUANTITIES

This section proposes a three-step estimation routine for the unknown quantities: the setpoint, the local model, and the domain of attraction. The estimation phase is performed offline, hence LAS property is not required. Then, the process can rely on a sufficiently precise simulator. Even if the estimation is not perfect, the global policy is still guaranteed to be locally stabilizing, since it can withstand bounded perturbations without loosing LAS [35]. First, given the desired goal state, it is necessary to estimate the setpoint constant action. Then, we need to estimate a linear model approximating the behavior of the system around that point. Since the model to be learned is linear, we can rely on classical system identification techniques [30]. However, we need to take care of the data collection procedure, due to the intrinsic nonlinearities of the environment. Finally, we have to approximate the size of the domain of attraction for the local stabilizing policy computed from the identified model.

3.1 Estimating the Setpoint Action

Setpoint optimal control tasks assume a desired goal state $x^\star$ is given. Nevertheless, without any model knowledge, we need to estimate which input $u^\star$ would make the pair $(x^\star, u^\star)$ a suitable setpoint for the closed-loop system. Mathematically, we look for $u^\star$ such that $x^\star = f(x^\star, u^\star)$. The estimated input $\widehat{u^\star}$ can be defined as the one minimizing the error between the goal state $x^\star$ and the successive one, which is observed once such input is applied. Hence, recalling that $x' = f(x,u)$, we aim at solving

$$\mathcal{L}(x,u) \triangleq \|x' - x\|^2, \quad \widehat{u^\star} = \arg\min_u \mathcal{L}(x^\star, u). \tag{5}$$

There are multiple possibilities to solve such an optimization problem, e.g., automatic differentiation tools. In our experiments, we use a genetic algorithm to find an initial estimate and successively refine it via gradient descent until the computed cost falls below a small threshold $\epsilon > 0$.

3.2 Estimating the Local Linear Model

A discrete-time linear deterministic dynamical system can be modeled as

$$\begin{aligned} x' &= Ax + Bu = Mz, \\ M &= \begin{pmatrix} A & B \end{pmatrix}, \quad z = (x^\top, u^\top)^\top. \end{aligned} \tag{6}$$

We aim at estimating the entries of matrices A and B such that (6) provides a good approximation of the unknown system dynamics $f(x,u)$ close to the setpoint $(x^\star, u^\star)$. Therefore, given a dataset of tuples (x, u, x'), the model estimation problem can be solved with least squares techniques. Yet, due to the nonlinear behavior of the system, some care has to be taken during the data collection in order to obtain a suitable local model. Since we look for a linearization of the system around the setpoint, we sample $N_e \in \mathbb{N}$ random control actions u_i from a normal distribution $u_i \sim \mathcal{N}(u^\star, \sigma^2) \quad i = 0, \dots, N_e - 1$, where $\sigma > 0$ is a sufficiently small standard deviation. We collect system trajectories by initializing its state in $x^\star$, i.e. $x_0 = x^\star$. Then, the control action u_0 is applied, the next state x'_0 is sampled and the triplet (x_0, u_0, x'_0) is stored. Before applying the successive action u_1, the trajectory is stopped and the system is initialized back in the goal state $x_1 = x^\star$ if the new state x'_0 lies outside of a box centered in the goal state $\mathcal{B}(x^\star) = \{x \in \mathbb{R}^n : |x^j - x^{\star j}| < b^j \quad j = 1, \dots, n\}$, where superscript j refers to the j^{th} component of the vector and $b \in \mathbb{R}^n$ is a vector of positive bounds. Otherwise, we directly apply the next input, i.e. $x_1 = x'_0$. The process is repeated until N_e state-action-state samples are collected. Once the data has been collected, the system matrix $\widehat{M}$ can be obtained by least square estimation [30]. Algorithm 1 presents the proposed procedure for linear model estimation given a setpoint and a vector of positive state bounds $b \in \mathbb{R}^n$.

3.3 Estimating the Domain of Attraction

The local solution $\boldsymbol{\pi}_{\text{loc}}$ in (3) can be computed with the knowledge of the linear system and the quadratic reward. Yet, since the size of the region where $\boldsymbol{\pi}_{\text{loc}}$ solves the task is still unknown, we propose an automatic procedure for obtaining a conservative estimation of its domain of attraction so that to set the shape of the blending functions h_1, h_2. There is a rich literature on domain of attraction estimation, see e.g., [8, 31]. Our approach exploits the solution to the Discrete-time Algebraic Riccati Equation (DARE). This solution is also used for the computation of the local policy. Moreover, the approach allows for a direct definition of suitable functions h_1, h_2.

Algorithm 1 Linearized system identification

Input: $x^\star, u^\star, b, N_e$;
Sample $u_i \sim \mathcal{N}(u^\star, \sigma^2)$, $i = 0, \dots, N_e - 1$;
Initialize the system in $x_0 = x^\star$;
Apply u_0 and observe $x_0' = f(x_0, u_0)$;
Store (x_0, u_0, x_0');
for i=1, …, N_e **do**
  if $|x_i^j - x^{\star j}| < b^j$ for all $j = 1, \dots, n$ **then**
    $x_1 = x_0'$;
  else
    Reset the system in $x_1 = x^\star$;
    Store (x_i, u_i, x_i');
  end if
end for
Compute $\widehat{M}$ with least squares;
Extract $\widehat{A}$ and $\widehat{B}$ as in (6);

Algorithm 2 Local domain of attraction estimation

Input: $x^\star, u^\star, \Delta_c, N_v, \delta$;
$(\widehat{c^\star}, c, \texttt{n\_fails}) \leftarrow (0, \Delta_c, 0)$;
while `n_fails` < `int`$(\delta N_v) + 1$ **do**
  Reset `n_fails`= 0;
  Sample x_i, $i = 1, \dots, N_v$ on the surface $\mathcal{E}(c)$;
  for i=1, …, N_v **do**
    Simulate $x' = f(x_i, \boldsymbol{\pi}_{\text{loc}}(x))$;
    if $V(x') - V(x_i) \geq 0$ **then**
      `n_fails` = `n_fails` + 1;
    end if
    if `n_fails` > `int`(δN_v) **then**
      $\widehat{c^\star} = c$; `break`;
    end if
  end for
  Update $c = c + \Delta_c$;
end while

It can be proved that LAS of nonlinear systems can be described by analyzing the linearized framework. In particular, the local asymptotic stability of a setpoint for the closed-loop system can be characterized via the existence of a non-negative real-valued function $V : \mathbb{R}^n \to \mathbb{R}$, denoted as Lyapunov function [5]. Such a function has to satisfy the following conditions

$$V(x^\star) = 0, \tag{7}$$

$$V(x) > 0 \quad \forall x \in \mathbb{R}^n - \{x^\star\}, \tag{8}$$

$$V(x') - V(x) < 0 \quad \forall x \in \mathbb{R}^n - \{x^\star\}. \tag{9}$$

Furthermore, since $\boldsymbol{\pi}_{\text{loc}}$ is the linear optimal policy, we can choose the optimal value function in the linear framework as a candidate local Lyapunov function in a quadratic form, namely of the form

$$V(x) = (x - x^\star)^\top P (x - x^\star), \tag{10}$$

where P is the positive definite solution of the DARE, see e.g. [4]. Note that conditions (7) and (8) are met by such a function by construction, hence it suffices to check whether (9) is satisfied at different points. Then, the domain of attraction is set as the states inside an ellipsoid described by (10) such that they all satisfy property (9). Practically, this size is approximately estimated by sampling $N_v \in \mathbb{N}$ random points on the surface of an increasingly bigger n-dimensional ellipsoids $\mathcal{E}(c_k) := \{x \in \mathbb{R}^n : V(x) \leq c_k\}$. Then, for each of these points, we check if the decrease condition (9) is satisfied by selecting actions with $\boldsymbol{\pi}_{\text{loc}}$ on the nonlinear system. We repeat the procedure for incremental values $c_{k+1} = c_k + \Delta_c$, $\Delta_c \in \mathbb{R}_{>0}$ until we find a sample violating the constraint (9). Then we set the estimated size $\widehat{c^\star} = c_k$. Note that, by continuity, a sufficiently fine sampling of the surface of the ellipsoid allows the generalization of the result to the complete surface. The same can be stated for sufficiently small Δ_c.

Once $\widehat{c^\star}$ is known, we can define a normalized Lyapunov function $v(x) = \frac{V(x)}{\widehat{c^\star}}$. This is useful to easily identify the states at the borders of the domain of attraction. Indeed, it is smaller than 1 inside the estimated domain, larger outside and 1 on the borders. Then, the saturation functions h_1, h_2 in (3) and (4) can be shaped as

$$h_1(x) = \tanh\left(s v(x)\right), \quad h_2(x) = \tanh\left(s v(x)^{3/2}\right), \tag{11}$$

where $s = \tanh^{-1}(\beta)$ and $\beta \in (0, 1)$ set the value of h_1,h_2 at points on the boundaries of the level set identified by $\widehat{c^\star}$. Note that equations (11) satisfy the requirements presented in [34] of being smooth, saturating, higher order functions with respect to $\boldsymbol{\pi}_{\text{loc}}$ and $\mathbf{Q}_{\text{loc}}$ respectively. In other words, they guarantee the control policy and the action value behave as the local quadratic optimal ones close to the setpoint, and as the learned one far from it. Moreover, the saturating action can be controlled by the value of β. If β is very small, at the borders of the estimated domain the control policy will still be dominated by the local solution $K^\star$. On the contrary, if $\beta \approx 1$ we let the learned component modify the solution even inside the estimated domain.

Algorithm 2 proposes the routine for estimating the domain of attraction of the local stabilizing policy. The method presented in this section provides an estimation of the stable region which may be very conservative. Indeed, it looks for the smaller invariant ellipsoid contained in the domain. A less conservative approach may be to set a threshold of allowed failures per level set. This can be controlled by setting a value $\delta \in [0, 1)$ different from zero in Algorithm 2. However, the bigger δ, the less confident we can be in the estimation.

3.4 The End-To-End Algorithm

In the previous sections, we showed how to select the setpoint input u^* given a desired setpoint goal x^*. We also presented how to identify a linearized model (6) of the system around such a setpoint to design a local stabilizing policy $\boldsymbol{\pi}_{\text{loc}}$. Finally, we suggested how to estimate its domain of attraction and select the functions h_1, h_2 in (3) accordingly. Therefore, the next step is to select an actor-critic DRL algorithm to learn the networks' parameters.

The complete algorithm runs in two consequent steps: unknown quantities offline estimation and policy learning. Once the local solution has been estimated, it is kept fixed. A LQR local stabilizing policy is built upon the learned linear model. Such a policy is robust with respect to bounded errors in the model and stability is guaranteed even if the learned model is not perfect [35]. Therefore, due to the robustness of the local solution $K^\star$, the control policy $\boldsymbol{\pi}^\theta$ is still

Algorithm 3 Locally asymptotically stabilizing actor critic DRL

1: Input: $x^\star, N_e, b, \Delta_c, N_v, Q_\gamma, R_\gamma, \gamma, \beta$;
2: Estimate $\widehat{u^\star}$ by solving (5);
3: Obtain $\widehat{A}$ and $\widehat{B}$ as in Sec. 3.2 with $\widehat{u^\star}$;
4: Compute $P, K^\star, H$ with $\widehat{A}$ and $\widehat{B}$ as in Appendix A ;
5: Set $\boldsymbol{\pi}_{\text{loc}}$ as in (3) with $\widehat{u^\star}$ and $K^\star$;
6: Obtain $\widehat{c^\star}$ as in Sec. 3.3;
7: Set h_1, h_2 as in (11) via $\widehat{c^\star}$;
8: Set $\boldsymbol{\pi}^\theta$ as in (3);
9: Set $\hat{\mathbf{Q}}^\phi_{\boldsymbol{\pi}}$ as in (4);
10: Train $\boldsymbol{\pi}^\theta$ and $\hat{\mathbf{Q}}^\phi_{\boldsymbol{\pi}}$ with DRL;

ensured to be locally stabilizing as long as the estimated model provides a good approximation of the linearized behavior. This reflects in the choice of bounds b defining the box around the setpoint used for the local model estimation in Section 3.2. Indeed, they should not describe too big of a region around the goal state $x^\star$ and may be dynamically changed if the resulting model does not fit sufficiently well the collected trajectories. Stability can be verified afterward by experiments. We also highlight that from system control theory it is known that intertwining the phases of model estimation and policy learning may cause the loss of the stability properties. Indeed, even switching between stable linear systems may cause instability [22, Section 2.4.1]. Therefore, once the local stabilizing policy is fixed, we only allow fine-tuning of the overall policy with the components learned by the DRL algorithm, which can still affect (slightly) the local behavior. Indeed, we stress that the overall policy perfectly matches the local LQR policy only at the setpoint.

The algorithm asks for the desired goal state $x^\star$ and some hyperparameters: the vector of bounds b, the number of estimation samples N_e, the number of samples per level set N_v, the step size Δ_c and the value β. It first estimates the setpoint input by solving the optimization problem (5). Then it estimates the linearized model. Once $\widehat{A}$ and $\widehat{B}$ are known, the local optimal policy $\boldsymbol{\pi}_{\text{loc}}$ is computed and its domain of attraction is estimated. Once the local problem has been solved, it sets the functions h_1 and h_2 as in (11) and trains the component $\boldsymbol{\mu}^\theta$. Given an actor-critic DRL algorithm, it outputs a near-optimal neural network policy with LAS guarantees.

Scaling-up and limitations. DRL benchmarks usually involve high-dimensional and complex environments. The proposed techniques directly extend to setpoint optimal control problems of big dimensions, as it translates to a standard DRL problem on a modified environment. Indeed, by fixing $\boldsymbol{\pi}_{\text{loc}}$, the environment under policy $\boldsymbol{\pi}^\theta$ and evolving according to

$$x' = f(x, \boldsymbol{\pi}^\theta(x))$$

can be seen a standard DRL problem on a modified environment under policy $\boldsymbol{\mu}^\theta$ and evolving according to

$$x' = \bar{f}(x, \boldsymbol{\mu}^\theta(x)) = f(x, \boldsymbol{\pi}_{\text{loc}} + h_1(x)(\boldsymbol{\mu}^\theta - \boldsymbol{\pi}_{\text{loc}})).$$

Moreover, it is known that LQR controllers can be computed for high-dimensional systems [2]. However, the more complex the system, the harder it may be to precisely define the setpoint $x^\star$. If the setpoint cannot be reached, then the proposed approach may fail to provide a good local model. As a consequence, if the robustness bounds of the global policy cannot withstand the error in the local model, LAS cannot be guaranteed. As such, different model estimation techniques may have to be used.

4 EXPERIMENTS

We conduct experiments on low-dimensional instances to show the need of adding stability properties even in relatively simple tasks. Indeed, stability has two main consequences: robustness and generalization. In particular, a stabilizing policy is able to complete the task on environments similar to the one it was designed for. In the following, we analyze four main aspects of the proposed solution. First, we compare performances of standard DRL algorithms against those from their locally asymptotically stabilizing variants (LAS policy). Next, we study the stability properties of both family of algorithms. Then, we compare the difference in domain of attraction size between the LAS solution and the classical local control-theoretic approach (LQR). Finally, we also present experiments showing that the LAS properties generalize well to different instances.

We train four different DRL algorithms, namely DDPG [23], PPO [27], TD3 [13] and SAC [18]. We compare the results with their LAS versions. We train each algorithm over four random seeds. The training is performed on a single NVidia GTX Titan X with 12 GB RAM. The standard formulation of the algorithms comes from Stable Baselines 3 [26] and it is modified for obtaining the LAS design. For the estimation routine we set $N_e = 3 \times 10^5$ and $N_v = 5 \times 10^3$. Table 5 in Appendix A presents the main hyperparameters used for each algorithm and control task. Most of them are provided by the Stable Baselines 3 library. The hyperparameters are not optimized. The main goal is to show that LAS is a valuable property, especially in non-optimal conditions. All the hyperparameters that are not shown in the table are also set as the default ones. For all the algorithms, we use a discount factor $\gamma = 0.99$, in order to provide a sufficiently big virtual time horizon to the optimization process.

4.1 Control Tasks

We set up three different environments of increasing complexity for evaluating algorithms.

Pendulum swing-up (PSU). A frictionless pendulum has to be stabilized at a small angle from the unstable vertical position. The input torque is limited in amplitude. The environment follows the same dynamics as OpenAI Gym [7] pendulum swing-up. The state vector is composed of the error angle with respect to the top vertical position and its rate of change ($n = 2$). The goal state is $x^\star = (\frac{10}{180}\pi, 0)^\top$ and the initial condition is $x_0 = (\pi, 0)^\top$. Table 4 in Appendix A presents the environment parameters.

Inverted pendulum swing-up (IPSU). A pendulum linked to a cart has to be stabilized in its unstable vertical position starting from the resting downward position. The environment is built using Pybullet [12]. The cart can slide on a rail of limited length and the input force on the cart is bounded. The cart is also required to reach the middle of the rail. The state vector is composed of the distance between the cart and the center of the rail, the error angle with respect to the top vertical position of the pole, the cart velocity and the angle rate of change ($n = 4$). The goal state is $x^\star = (0, 0, 0, 0)^\top$

and the initial condition is $x_0 = (0, \pi, 0, 0)^\top$. Table 7 in Appendix A presents the environment parameters.

Double inverted pendulum swing-up (DIPSU). The IPSU task has to be solved with a 2-link pendulum. The state vector is composed of the distance between the cart and the center of the rail, the error angle with respect to the top vertical position of the first pole, the error angle of the second pole with respect to the first pole's direction, the cart velocity and the angles rates of change ($n = 6$). The goal is to stabilize the system at $x^\star = (0, 0, 0, 0, 0, 0)^\top$ and the initial condition is $x_0 = (0, \pi, 0, 0, 0, 0)^\top$. Table 6 in Appendix A presents the environment parameters.

4.2 Performances

The objective is to show that the proposed algorithm does not deteriorate performances of standard DRL solutions. The difference in performances can be clearly seen by looking at the episode return during training. Following [34], the global discounted objective is designed starting from a local undiscounted one. This ensures the local solution is stabilizing. Moreover, it allows standard DRL algorithm trained on such an objective to possibly converge to a locally stabilizing policy. For the local component, we use undiscounted quadratic costs described by matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}$, where Q is diagonal and defined as $Q = \mathbf{I}_n \odot (\mathbf{1}_n W)$, being $\mathbf{I}_n$ the identity matrix of dimension n, $\mathbf{1}_n \in \mathbb{R}^n$ a column vector of ones, $W \in \mathbb{R}^n$ a row vector of weights and $\odot$ the Hadamard product. The global cost matrices are defined as $Q_\gamma = \gamma Q + (1 - \gamma)\bar{P}$, $R_\gamma = \gamma R$, where $\bar{P}$ is the solution to the DARE for the undiscounted problem. Given that typical DRL algorithms solve a maximization problem, we invert the sign of the instantaneous cost and of the local component of the critic. For the PSU task we use matrices $W = \begin{pmatrix} 1 & 0.1 \end{pmatrix}$, $R = 0.001$. For the IPSU task we set $W = \begin{pmatrix} 0.1 & 0.1 & 0.01 & 0.01 \end{pmatrix}$ $R = 0.01$. Finally, the optimization objective of DIPSU is described by matrices $W = \begin{pmatrix} 0.001 & 0.3 & 0.3 & 0.001 & 0.01 & 0.01 \end{pmatrix}$, $R = 0.0001$.

From Table 1 and Fig. 1, we see that the LAS variants perform comparably to their standard versions. As expected, the approach embeds the local exact solution in the neural policy, thus it does not impact the performances once the $\boldsymbol{\pi}_{\text{loc}}$ component dominates close to the setpoint. Moreover, in some instances the LAS policy performs even better than the native algorithm. This is probably due to the fact that the local component guides the exploration towards stabilizing actions. Since quadratic costs are proportional to the norm of the error, stabilizing actions encourage the reduction of the instantaneous cost.

4.3 Local Asymptotic Stability

We assess the stability properties of both types of policies, *i.e.,* LAS and standard policies. To do so, we test them in corrupted versions of the training environments. Here, unseen measurement noise, external disturbances, and parameter mismatch perturb the control task. Each corrupted task can be modeled as $x' = f(x, \text{sat}(\psi(x+w))+d)$, where $\psi : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary policy (e.g. $\boldsymbol{\pi}^\theta$), $\text{sat} : \mathbb{R} \to \mathbb{R}$ is a hard saturation function clipping the control input, w is a random noise sample at time t and d is an external input disturbance. The metric for assessing stability is defined as the maximum error at steady state, i.e. after some time we start looking at the maximum norm of the error with respect to the goal state. This "stability cost" can be described as $s_\psi = \max_{t \in [t_0, T]} \|x - x^\star\|$ where $T \in \mathbb{N}$ represents the maximum episode length in time steps and $t_0 \in [0, T)$ is the instant at which we start evaluating the stability properties.

Table 3 presents the stability cost of each algorithm after training. It shows that LAS policies outperform the standard version in the local context, i.e., when starting sufficiently close to the setpoint. Note also the reliability of such a local behavior, as shown by the small variation of the stability cost over different instances. As expected, the local stabilizing policy guarantees LAS and consequently small error norms. Differently, standard algorithms may not converge to a stabilizing policy. The smoothed evolution of the stability cost during training is shown in Fig. 2. Again, we note from the plots of Fig. 2, that the standard algorithms are not capable of locally stabilizing the system in the corrupted environments, due to low diversity of the dataset. On the contrary, the LAS version shows a small error norm even with the untrained parameters, thanks to the local knowledge. We also emphasize that the local solution maintains a small error norm if the initial condition lies outside the domain of attraction of $\boldsymbol{\pi}_{\text{loc}}$ but the learned component manages to steer the system sufficiently close to the setpoint before t_0, as shown by Table 3 and Fig. 2 (left). This does not always happen in Fig. 2 (center) and Fig. 2 (right), and the LAS variants behave similarly to their native versions. Nevertheless, the local stabilizing policy seems to improve the stability cost also in the global context. Finally, we stress that LAS is preserved even if the algorithm does not reach a satisfactory solution, as for PPO in Fig. 1 (right) and Fig. 2 (right).

4.4 Generalization

This subsection addresses whether standard DRL algorithms can achieve LAS with an increased amount of data. We focus on the simplest task, the PSU. The standard algorithms are trained on three instances with different masses, while the LAS version is trained on a single one with the default mass $m = 1$. Hence, we are providing a richer dataset to the nominal algorithms to show that local generalization is obtained with small data overhead with respect to standard techniques. We set the corrupted environment mass to $m_{corr} = 1.2$ and we explore two training situations. First, we present the results of experiments where the m_{corr} is between the smallest and biggest values experienced by the algorithms, see Fig. 3 (left). Then, we show the results for the case when the corrupted mass lies outside these boundaries, see Fig. 3 (right). We note that in both cases LAS is not guaranteed by the standard algorithms, even when a higher amount of information is provided. This is particularly evident when the corrupted mass is bigger then the maximum experienced one, see Fig. 3 (right). Nevertheless, we can notice an improvement with respect to the single environment scenario, especially for PPO. On the other hand, the LAS approach ensures satisfactory generalization capabilities over a bounded, continuous set of parameters' variations around the nominal ones, even if trained on a single environment. The continuity of such a set and the reduced amount of required data further motivate the interest in the LAS approaches.

4.5 Domain of Attraction

We evaluate the size of the domain of attraction of the composite policy and compare it to the linear control-theoretic solution (LQR). Hence, we uniformly sample $N_i \in \mathbb{N}$ initial conditions inside a box

Table 1: Mean evaluation rewards and standard deviations across ten trials (the higher the better).

	PPO		SAC	
	LAS	**Standard**	**LAS**	**Standard**
PSU	$-160,2 \pm 97,0$	$-162,4 \pm 91,9$	$-133,7 \pm 62,4$	$-133,9 \pm 77,5$
IPSU	$-142,4 \pm 35,8$	$-104,8 \pm 21,7$	$-92,3 \pm 1,8$	$-102,0 \pm 8,4$
DIPSU	$-579,1 \pm 156,3$	$-1706,9 \pm 481,2$	$-245,4 \pm 5,8$	$-447,1 \pm 5,9$

	TD3		DDPG	
	LAS	**Standard**	**LAS**	**Standard**
PSU	$-146,4 \pm 72,1$	$-135,5 \pm 101,4$	$-135,1 \pm 63,9$	$-156,8 \pm 77,3$
IPSU	$-99,0 \pm 7,3$	$-99,7,0 \pm 3,7$	$-120,4 \pm 29,0$	$-155,6 \pm 28,1$
DIPSU	$-251,3 \pm 5,7$	$-255,3 \pm 7,8$	$-487,4 \pm 287,4$	$-485,1 \pm 196,0$

Table 2: Success rate over $N_i = 10000$ initial conditions. The success threshold: $\rho = 0.001$. Initial condition bounds b_0 for the different tasks: $[\pi, 8]$ (PSU), $[1, \pi, 10, 3\pi]$ (IPSU), $[1, \pi, \pi, 10, 3\pi, 3\pi]$ (DIPSU).

	PSU	**IPSU**	**DIPSU**
LQR	0.6432	0.0856	0.0151
LAS-DDPG	1.0	0.8028	0.3362
LAS-PPO	0.9997	0.9987	0.0682
LAS-TD3	1.0	0.9454	0.4012
LAS-SAC	1.0	0.9501	0.4646

Table 3: Maximum steady-state error norm in corrupted environments over ten trials (the lower the better). Subscript '1' indicates the system is initiated in $x^\star$. Subscript '2' indicates the system is initiated in x_0.

	PPO		SAC		TD3		DDPG	
	LAS	**Standard**	**LAS**	**Standard**	**LAS**	**Standard**	**LAS**	**Standard**
PSU_1	$0,18 \pm 0,02$	$8,63 \pm 0.2$	$0,1 \pm 0,01$	$0,25 \pm 0,02$	$0,1 \pm 0,01$	$7,37 \pm 0,06$	$0,09 \pm 0,01$	$0,25 \pm 0,01$
PSU_2	$0,17 \pm 0,02$	$8,64 \pm 0,1$	$0,1 \pm 0,01$	$0,27 \pm 0,03$	$0,1 \pm 0,01$	$7,42 \pm 0,03$	$0,09 \pm 0,01$	$0,24 \pm 0,01$
$IPSU_1$	$0,69 \pm 0,05$	$1,95 \pm 2,76$	$0,80 \pm 0,12$	$1,07 \pm 0,25$	$0,76 \pm 0,06$	$0,92 \pm 0,10$	$0,66 \pm 0,04$	$3,27 \pm 2,08$
$IPSU_2$	$3,58 \pm 3,99$	$7,16 \pm 0,20$	$0,81 \pm 0,08$	$1,18 \pm 0,39$	$0,86 \pm 0,17$	$2,46 \pm 3,32$	$0,65 \pm 0,05$	$5,15 \pm 3,09$
$DIPSU_1$	$2,28 \pm 0,11$	$11,88 \pm 4,73$	$2,29 \pm 0,27$	$7,04 \pm 4,07$	$2,37 \pm 0,19$	$9,06 \pm 0,13$	$2,40 \pm 0,12$	$3,27 \pm 0,25$
$DIPSU_2$	$6,58 \pm 5,55$	$15,17 \pm 8,77$	$2,47 \pm 0,12$	$3,92 \pm 3,60$	$2,24 \pm 0,16$	$8,97 \pm 0,07$	$2,83 \pm 0,97$	$3,26 \pm 1,60$

$\mathcal{B}(0) = \{x \in \mathbb{R}^n : |x^j| < b_0^j \quad j = 1, \ldots, n\}$, where superscript j refers to the j^{th} component of the vector and $b_i \in \mathbb{R}^n$ is a vector of positive bounds. Then, we run experiments in the uncorrupted environment, once with the trained locally stable neural policy and once with the LQR. Given an initial condition x and an episode length $t_e > 0$, the experiment is considered a success if the error between the final state and the goal state is smaller than a small threshold $\rho > 0$. An evaluation metric $e \in [0, 1]$ is given as a rate between the number of successes and the number of initial conditions, namely $e = \frac{\texttt{n\_successes}}{N_i} = \frac{1}{N_i} \sum_{k=0}^{N_i} \eta(x_{t_e}^k)$, where the k superscript identifies the experiment initialized with the k^{th} initial condition and $\eta : \mathbb{R}^n \to \mathbb{R}$ is defined as $\eta(x) = 1$ if $\|x - x^\star\| \leq \rho$ and $\eta(x) = 0$, otherwise. Table 2 shows that for each task the neural solution presents a bigger rate of success than the local LQR. Hence, the size of the domain of attraction is improved by the learned component in comparison to the control-theoretic linearization technique. It shows we can safely embed and train a neural component into classical control-theoretic solutions without impairing their local properties. Moreover, it highlights the effectiveness of modern learning approaches in improving established control methods.

5 CONCLUSIONS

In this paper, we presented an end-to-end algorithm embedding control-theoretic stability in actor-critic algorithms for setpoint optimal control tasks. The experiments show that the modified algorithms perform comparably to their native version, while providing desirable local properties. Moreover, they present a bigger domain of attraction with respect to the classical control-theoretic solution they are based on. Extension of this work to more general control problems, such as tracking of time-varying references, is currently under investigation. Similarly, improvements in the estimation routine are part of ongoing research.

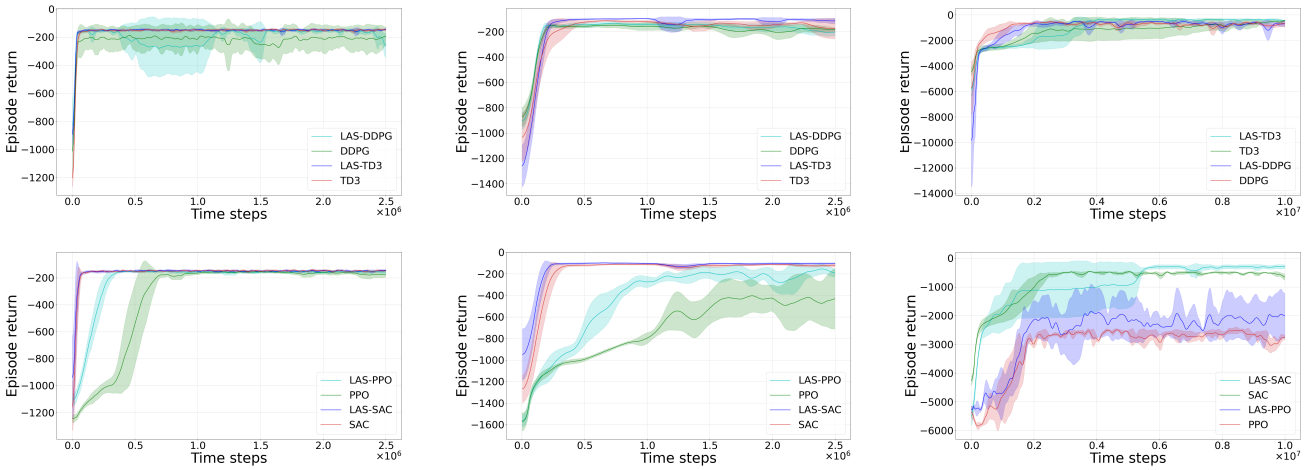

Figure 1: Training return: PSU (left), IPSU (center), DIPSU (right).

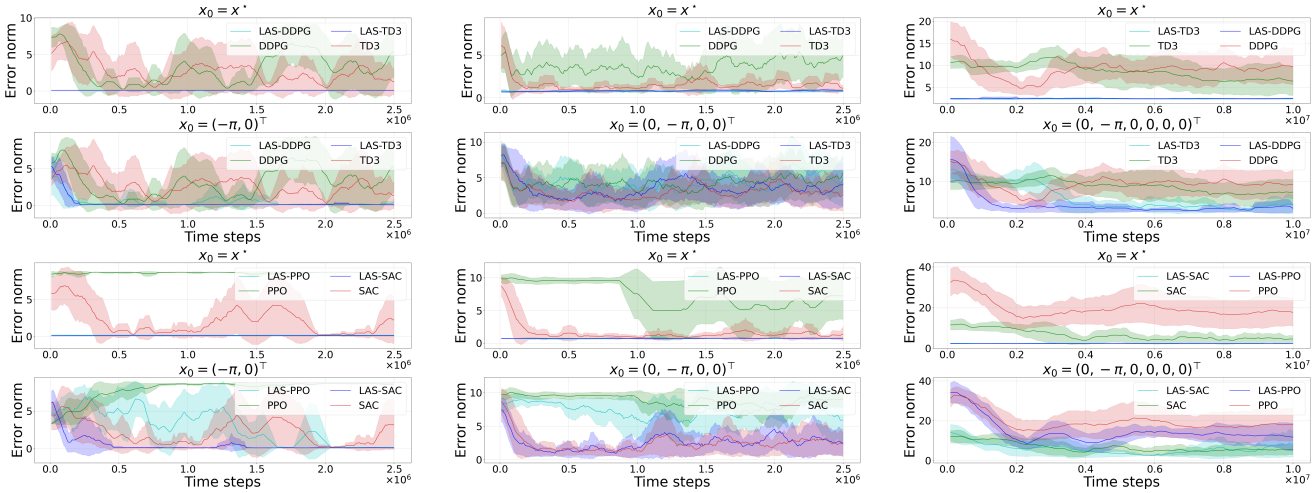

Figure 2: Stability score during training: PSU (left), IPSU (center), DIPSU (right). Row 1 and 3: local. Row 2 and 4: global.

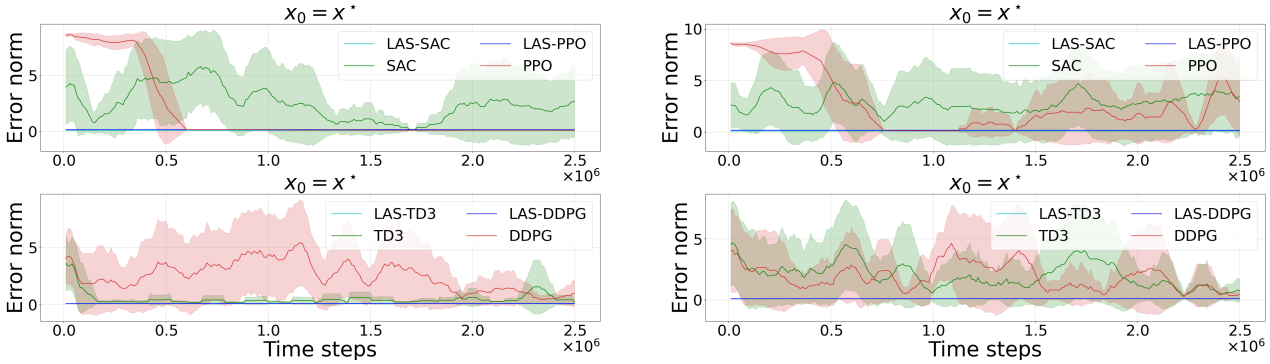

Figure 3: Local stability over multiple instances of PSU. Standard algorithms training masses set $m_{train} = [0.7, 1, 1.3]$ (left), $m_{train} = [0.9, 1, 1.1]$ (right).

REFERENCES

[1] V. Andrieu, B. Jayawardhana, and L. Praly. 2016. Transverse exponential stability and applications. *IEEE Trans. Automat. Control* (2016).

[2] P. Benner. 2004. Solving large-scale control problems. *IEEE Control Systems Magazine* 24, 1 (2004), 44–59.

[3] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. 2017. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551* (2017).

[4] D. P. Bertsekas. 2018. Dynamic Programming and Optimal Control 4th Edition, Volume II Chapter 4 Noncontractive Total Cost Problems. (2018).

[5] N. Bof, R. Carli, and L. Schenato. 2018. Lyapunov theory for discrete time systems. *arXiv preprint arXiv:1809.05289* (2018).

[6] S. J Bradtke. 1993. Reinforcement learning applied to linear quadratic regulation. In *Advances in neural information processing systems*. 295–302.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[8] G. Chesi. 2011. *Domain of attraction: analysis and control via SOS programming*. Vol. 20. Springer.

[9] J. Choi, F. Castaneda, C. J. Tomlin, and K. Sreenath. 2020. Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions. *arXiv preprint arXiv:2004.07584* (2020).

[10] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. 2018. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708* (2018).

[11] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh. 2019. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031* (2019).

[12] E. Coumans and Y. Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

[13] S. Fujimoto, H. Hoof, and D. Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 1587–1596.

[14] L. Furieri, C. L. Galimberti, and G. Ferrari-Trecate. 2022. Neural System Level Synthesis: Learning over All Stabilizing Policies for Nonlinear Systems. *arXiv preprint arXiv:2203.11812* (2022).

[15] L. Furieri, Y. Zheng, and M. Kamgarpour. 2020. Learning the globally optimal distributed LQ regulator. In *Learning for Dynamics and Control*. PMLR, 287–297.

[16] J. Garcıa and F. Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[17] F. Gu, H. Yin, L. El Ghaoui, M. Arcak, P. Seiler, and M. Jin. 2022. Recurrent neural network controllers synthesis with stability guarantees for partially observed systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 5385–5394.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*. PMLR, 1861–1870.

[19] M. Han, L. Zhang, J. Wang, and W. Pan. 2020. Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters* 5, 4 (2020), 6217–6224. https://doi.org/10.1109/LRA.2020.3011351

[20] S. Khadka, S. Majumdar, T. Nassar, Z. Dwiel, E. Tumer, S. Miret, Y. Liu, and K. Tumer. 2019. Collaborative evolutionary reinforcement learning. In *International Conference on Machine Learning*.

[21] H. K. Khalil. 2002. *Nonlynear systems* (3rd ed ed.).

[22] D. Liberzon. 2003. *Switching in systems and control*. Vol. 190. Springer.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[25] R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz. 2016. Stability analysis of discrete-time infinite-horizon optimal control with discounted cost. *IEEE Trans. Automat. Control* 62, 6 (2016), 2736–2749.

[26] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. 2019. Stable Baselines3. https://github.com/DLR-RM/stable-baselines3.

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

[28] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[29] E. D. Sontag. 2008. Input to state stability: Basic concepts and results. In *Nonlinear and optimal control theory*.

[30] T. Söderström and P. Stoica. 1989. *System Identification*.

[31] G. Valmorbida and J. Anderson. 2017. Region of attraction estimation using invariant sets and rational Lyapunov functions. *Automatica* 75 (2017), 37–45.

[32] E. Zergeroglu, W.E. Dixon, D.M. Dawson, S. Jaffar, and M.W. Hannan. 1998. Lyapunov-based set-point control of the Acrobot. In *Proceedings of the 1998 IEEE International Conference on Control Applications*.

[33] N. Zhang and N. Capel. 2021. LEOC: A Principled Method in Integrating Reinforcement Learning and Classical Control Theory. In *Learning for Dynamics and Control*. PMLR.

[34] S. Zoboli, V. Andrieu, D. Astolfi, G. Casadei, Jilles S. Dibangoye, and Madiha Nadri. 2021. Reinforcement learning policies with local LQR guarantees for nonlinear discrete-time systems. *2021 IEEE 60th Conference on Decision and Control (CDC)* (2021).

[35] S. Zoboli, D. Astolfi, and V. Andrieu. 2022. Total stability and integral action for discrete-time nonlinear systems. *arXiv preprint arXiv:2210.03450* (2022).

A APPENDIX

Infinite horizon discounted LQR. Given a deterministic discrete-time linear system as in (6) and a quadratic γ-discounted cost function (1), the optimal policy is (see e.g., [4, Chapter 4.3])

$$\pi^\star(x) = K^\star(x - x^\star), \quad K^\star = -\gamma(R_\gamma + \gamma B^\top P B)^{-1} B^\top P A, \quad (12)$$

where $K^\star$ is the discounted optimal gain and P is the unique positive solution of the discounted Discrete-time Algebraic Riccati Equation (DARE)

$$P = Q_\gamma + \gamma A^\top (P - \gamma P B (R_\gamma + \gamma B^\top P B)^{-1} B^\top P) A. \quad (13)$$

From [6, Section 3], the value functions for the discounted LQR problem under the optimal controller are

$$\mathbf{J}^\star(x) = x^\top P x, \qquad \mathbf{Q}^\star(x, u) = z^\top H z, \quad (14)$$

with $z := \mathrm{col}(x, u) \in \mathbb{R}^{n+m}$ and H given by

$$H = \begin{pmatrix} Q_\gamma + \gamma A^\top P A & \gamma A^\top P B \\ \gamma B^\top P A & R_\gamma + \gamma B^\top P B \end{pmatrix}.$$

Table 4: Pendulum swing-up environment parameters

Parameters	Training	Corrupted
pole mass m	1	1.2
pole length l	1	1
gravity acceleration g	9.81	9.81
episode max length T	200	1000
input bounds	[-2,2]	[-2,2]
noise w	0	$w \sim \mathcal{N}(0, 0.1)$
disturbance d	0	$0.2\sin(\frac{2\pi}{100}t)$
steady-state threshold t_0	-	500

Table 6: Inverted pendulum swing-up environment parameters

Parameters	Training	Corrupted
cart mass m_c	10.47	10.47
pole mass m_p	5	6.53
pole length l_p	0.6	0.8
rail bounds l_r	[-1,1]	[-1,1]
episode max length T	1000	1000
input bounds	[-100,100]	[-100,100]
noise w	0	$w \sim \mathcal{N}(0, 0.173)$
disturbance d	0	$20\sin(\frac{2\pi}{50}t)$
steady-state threshold t_0	-	500

Table 7: Double inverted pendulum swing-up environment parameters

Parameters	Training	Corrupted
cart mass m_c	10	10
first pole mass m_{p1}	1	1
second pole mass m_{p2}	1	1.2
first pole length l_{p1}	0.6	0.6
second pole length l_{p2}	0.6	0.7
rail bounds l_r	[-2,2]	[-2,2]
episode max length T	1000	2000
input bounds	[-200,200]	[-200,200]
noise w	0	$w \sim \mathcal{N}(0, 0.173)$
disturbance d	0	$20\sin(\frac{2\pi}{100}t)$
steady-state threshold t_0	-	1500

Table 5: Hyperparameters. lr: learning rate, af: activation function, NN: hidden layer sizes

	PSU			IPSU			DIPSU		
	lr	af	NN	lr	af	NN	lr	af	NN
DDPG	0.001	ReLU	400,300	0.001	ReLU	400,300	0.0001	ReLU	400,300
LAS-DDPG	0.001	ReLU	400,300	0.001	ReLU	400,300	0.0001	Tanh	400,300
PPO	0.003	Tanh	64,64	0.00025	Tanh	256,256	0.0001	Tanh	64,64
LAS-PPO	0.003	Tanh	64,64	0.00025	Tanh	256,256	0.0001	Tanh	64,64
TD3	0.001	ReLU	400,300	0.001	ReLU	400,300	0.0006	ReLU	400,300
LAS-TD3	0.001	ReLU	400,300	0.001	ReLU	400,300	0.0002	Tanh	256,256
SAC	0.001	ReLU	256,256	0.001	ReLU	256,256	0.001	ReLU	256,256
LAS-SAC	0.001	ReLU	256,256	0.001	ReLU	256,256	0.0001	Tanh	256,256