Off Policy Lyapunov Stability in Reinforcement Learning

Sarvan Gill
Department of Mechanical Engineering
University of Victoria, Canada
sarvan13@uvic.ca

Daniela Constantiescu
Department of Mechanical Engineering
University of Victoria, Canada
danielac@uvic.ca

Abstract: Traditional reinforcement learning lacks the ability to provide stability guarantees. More recent algorithms learn Lyapunov functions alongside the control policies to ensure stable learning. However, the current self-learned Lyapunov functions are sample inefficient due to their on-policy nature. This paper introduces a method for learning Lyapunov functions off-policy and incorporates the proposed off-policy Lyapunov function into the Soft Actor Critic and Proximal Policy Optimization algorithms to provide them with a data efficient stability certificate. Simulations of an inverted pendulum and a quadrotor illustrate the improved performance of the two algorithms when endowed with the proposed off-policy Lyapunov function.

Keywords: Reinforcement Learning, Control, Stability, Lyapunov

1 Introduction

Deep Reinforcement Learning (DRL) is emerging as a common robot control strategy because of its many recent promising results in challenging control tasks for systems with strongly non-linear dynamics and high dimensional state spaces, where classical control methods may struggle [1]. Learning from experience is a pillar of Reinforcement Learning (RL) and an agent's success is directly tied to the experience it learns from [2]. Given that it can be unsafe for a robot to collect trial and error samples of experience in the real world, safety and sample efficiency are important considerations for RL in robotics.

Stability is prerequisite for the safety of controlled systems. Given that unstable systems are unpredictable and can be dangerous, practical applications that require reliable and safe robots demand that the robots be guaranteed stable during task execution. Early sample-based RL techniques cannot certify stability. More recent RL methods aim to incorporate Lyapunov stability mechanisms into robot learning [3], generally by computing a Lyapunov function for the task error of the robot in closed-loop with the RL agent. While the existence of a Lyapunov function is sufficient for stable learning, a fundamental challenge to the stability analysis of RL for robotics arises from the fact that no systematic approach exists for determining Lyapunov functions for non-linear systems. RL research has tackled this challenge for some time [4].

In model-based RL, Lyapunov functions that use a model of the system dynamics guarantee stability directly [5], while control barrier functions ensure it through certifying safety [6, 7]. For control affine systems with known dynamics, solving for a control Lyapunov function leads to a list of permissible stabilizing controls [8, 9]. In model-free RL, a backup safe controller can be included to guarantee stability [10], including during online training [11]. Otherwise, model-free RL must turn to sample-based stability guarantees, generally by starting with a candidate Lyapunov function and then finding a control policy that makes the candidate Lyapunov [12, 13, 14]. In this approach, the value function often serves as the candidate and the reward function must be reshaped into a cost whose minimum has a value of zero at the equilibrium of the system. Alternatively, better

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

performance can be achieved by learning a neural Lyapunov function through a Lyapunov risk loss function which penalizes the neural network for any violations of the Lyapunov conditions [15]. A self-learned neural Lyapunov function neither requires changes to the reward function nor restricts the candidate to being the RL value function.

Whereas a candidate Lyapunov function has the advantage of being able to use off-policy data to increase sample efficiency during training, learning a neural Lyapunov function directly dispenses with the overhead in creating a cost function and can produce better results [15, 16]. However, to the authors' best knowledge, direct learning of a neural Lyapunov function is currently limited to learning off on-policy data. This paper introduces a novel method to learn a Lyapunov function directly using either on-policy or off-policy data. The proposed method extends [15] to account for off-policy data and includes an additional hyper parameter that controls a minimum rate of decay of the Lyapunov function. In essence, the paper:

- provides a framework that can learn Lyapunov functions off policy;
- demonstrates how the off-policy Lyapunov functions can guide state-of-the-art RL algorithms to learn stable policies;
- illustrates that, compared to other stable RL methods, the proposed framework can increase sample efficiency without performance sacrifices.

2 Preliminaries

This section briefly recalls the definitions needed to introduce the proposed off-policy Lyapunov functions in Section 3.

Consider the closed-loop system

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), u(\mathbf{x}(t))), \mathbf{x}_0 = \mathbf{x}(0), \tag{1}$$

with state $\mathbf{x}(t) \in \mathcal{X} \subseteq \mathbb{R}^n$, control signal $u(\mathbf{x}(t)) : \mathcal{X} \to \mathbb{R}^m$, and continuous non-linear dynamics $f : \mathcal{X} \to \mathbb{R}^n$.

Lyapunov Stability

An equilibrium state $\mathbf{x}_e \in \mathcal{X}$ of the closed-loop system (1) is Lyapunov stable if for every $\epsilon \in \mathbb{R}_{>0}$ there exists $\delta \in \mathbb{R}_{>0}$ such that $||\mathbf{x}_0 - \mathbf{x}_e|| < \delta$ implies $||\mathbf{x}(\mathbf{t}) - \mathbf{x}_e|| < \epsilon$ for all $t > 0$. The equilibrium $\mathbf{x}_e$ is asymptotically stable if it is Lyapunov stable and there exists $\delta \in \mathbb{R}_{>0}$ such that $||\mathbf{x}_0 - \mathbf{x}_e|| < \delta$ implies $\lim_{t\to\infty} \mathbf{x}(\mathbf{t}) = \mathbf{x_e}$.

Lyapunov Stability Criterion

An equilibrium state $\mathbf{x}_e \in \mathcal{X}$ of the closed-loop system (1) is Lyapunov stable if the system admits a Lyapunov function, that is, a positive semi-definite function $L : \mathcal{X} \to \mathbb{R}_{\geq 0}$ whose value is zero at the equilibrium, $L(\mathbf{x}_e) = 0$, and is positive at every other state, $L(\mathbf{x}) > 0$ for all $\mathbf{x} \neq \mathbf{x}_e$, and whose Lie derivative is non-positive along all system trajectories, $\mathcal{L}_f L(\mathbf{x}) = \nabla L \cdot f(\mathbf{x}(t), u(\mathbf{x}(t))) \leq 0$. The equilibrium $\mathbf{x}_e$ is asymptotically stable if the Lie derivative of the Lyapunov function is strictly negative, $\mathcal{L}_f L(\mathbf{x}) < 0$.

Neural Lyapunov Functions

While the existence of a Lyapunov function L is sufficient to certify the stability of the equilibrium $\mathbf{x}_e$, classical control theory offers no analytical method for deriving such a Lyapunov function. However, recent work [17, 18] has shown that parametrized neural networks can estimate Lyapunov functions. For control tasks, effective and certifiable Lyapunov functions can be learned by training a neural Lyapunov function L_θ to minimize the Lyapunov risk J_{L_θ} over an on-policy dataset $\mathcal{B}$ [17]:

$$J_{L_\theta} = \mathbb{E}_{\mathcal{B}} \left[\max(0, -L_\theta(\mathbf{x})) + \max(0, \mathcal{L}_f L_\theta(\mathbf{x})) + L_\theta(\mathbf{x}_e)^2 \right] \tag{2}$$

Reinforcement Learning

This paper considers a dynamical system that can be modeled by a Markov Decision process (MDP). Namely, the system is defined by the interaction of an RL agent with an environment. As the result of an action $a_t \in \mathcal{A} \subset \mathbb{R}^m$ taken by the agent at time t, the state $s_t \in \mathcal{S} \subset \mathbb{R}^n$ of the system changes to a new state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$. These transition probabilities define the system dynamics. Upon associating a reward function $R(s_t, a_t, s_{t+1})$ with the transition from s_t to s_{t+1} under a_t, the RL agent aims to learn a policy $\pi(a_t|s_t)$ that maximizes the reward it receives, typically parameterized as a neural network. In model-free RL, the transition probabilities and the reward function are not visible to the agent. Instead, the environment provides the appropriate signals, i.e., r_t and s_{t+1} are provided to the agent after taking an action a_t in state s_t. The RL agent seeks to maximize the total expected return, $J = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where γ is a discount factor that weights the value of future returns. This paper considers robotic systems in closed loop with an RL agent whose goal is to drive the robot to a goal state $s_G \in \mathcal{S}$.

Lyapunov Control in Reinforcement Learning

For an MDP, the Lie derivative of the Lyapunov function, $\mathcal{L}_f L_\theta$, can be modeled using the following finite difference derivative [15]:

$$\mathcal{L}_{f,\Delta t} L = \frac{L(s') - L(s)}{\Delta t}, \tag{3}$$

where s and s' are the two consecutive states with time difference Δt. When the data is sampled from the same policy, the finite difference of the Lyapunov function approximates its Lie derivative effectively. For on-policy data, the RL agent can self-learn Lyapunov functions by replacing the Lie derivative $\mathcal{L}_f L_\theta(\mathbf{x})$ in (2) with $\mathcal{L}_{f,\Delta t} L$ in (3), resulting in the following Lyapunov risk [15]:

$$J_{L_\theta} = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}} \left[\max(0, -L_\theta(s) + \max(0, \mathcal{L}_{f,\Delta t} L_\theta) + L_\theta(s)^2\right] \tag{4}$$

The Lyapunov risk (4) can then be used to train a Lyapunov function alongside an RL policy.

3 Learning Off-Policy Lyapunov Functions

This section proposes to extend the Lyapunov risk (4) to account for off-policy data. The inspiration comes from [13], where the RL action-value function $Q(s, a)$ serves as a Lyapunov candidate and the Lyapunov function is evaluated as the expectation over the actions under the current policy.

Instead of using a predetermined candidate, an off-policy self-learned Lyapunov function can be determined in two steps. In a first step, similarly to [13], the RL agent learns a neural Lyapunov function that depends both on the state and on the action. In a second step, the agent uses the expectation over the actions under the current policy to verify the Lyapunov conditions.

Formally, the agent learns a neural Lyapunov function $L_\eta(s, a)$ which is trained on the updated Lyapunov risk (5) with the redefined finite difference Lie derivative (6) calculated over an off-policy dataset D:

$$J_L(\eta) = \mathbb{E}_{(s,a,r,s')\sim D} \big[\max(0, -L_\eta(s, a)) + \max(0, \mathcal{L}_{f,\Delta t} L_\eta)\big] + L_\eta(s_G, \pi(s_G))^2 \tag{5}$$

$$\mathcal{L}_{f,\Delta t} L_\eta = \frac{L_\eta(s', \pi(s')) - L(s, a)}{\Delta t} \tag{6}$$

The key differences between (5) and (4) are the Lie derivative and the equilibrium value. The Lie derivative in (6) is explicitly dependent on the current policy, as in [13], where the decreasing condition serves to transform the RL action-value function into a Lyapunov function. This explicit dependence on the policy is necessary for off-policy learning as the data is no longer sampled under the same policy. Intuitively, the Lie derivative in (6) is now dependent on the action that the current

policy would take if it ended up in some state s'. Furthermore, the minimum of (5) also depends on the action taken there under the current policy.

To verify that the function learned by the risk (5) is a Lyapunov function as required by the Lyapunov stability criterion, consider the expectation of $L_\eta(s, a)$:

$$L_\eta(s) = \mathbb{E}_{a\sim\pi} L_\eta(s, a). \tag{7}$$

Note that $L_\eta(s, a) > 0$ and $L_\eta(s_G, \pi(s_G)) = 0$ together imply that $L_\eta(s) > 0$ and $L_\eta(s_G) = 0$. Furthermore, as shown in [13], $\mathcal{L}_{f,\Delta t} L_\eta(s, a)) < 0$ is sufficient for the Lie derivative of $L_\eta(s)$ to decrease along any system trajectory, $\mathcal{L}_{f,\Delta t} L_\eta(s)) < 0$.

3.1 Practical Changes

In practice, the RL agent learns the Lyapunov function with the help of a hyperparameter $\mu \in \mathbb{R}_{>0}$ which defines a minimum rate of decrease:

$$J_L(\eta) = \mathbb{E}_{(s,a,r,s')\sim D}\big[\max(0, -L_\eta(s, a)) + \max(0, \mathcal{L}_{f,\Delta t} L_\eta + \mu)\big] + L_\eta(s_G, \pi(s_G))^2 \tag{8}$$

and, thus, offers the ability to scale the changes in the Lyapunov function. While the shape of the function is sufficient to guarantee stability, a degree of control over its minimum rate of decrease can be used to impact the learning of the policy and the relative weight of the Lyapunov function in the policy update.

Imposing a minimum rate of decrease on the learned function makes it non-differentiable at the equilibrium of the system. The lack of a derivative at the equilibrium does not hinder the function from certifying stability because its Lie derivative can still be guaranteed negative everywhere but at the equilibrium. However, an important consideration is that the proposed loss function (8) cannot be zero by design, as it cannot decrease further by the required amount μ at the system equilibrium where it achieves its minimum. This issue can be side-stepped by using (8) to train the Lyapunov function and by using (5) to guarantee stability. Then, given the Lyapunov function learned by (8), the system is stable if (5) is satisfied.

3.2 Learning Stable Policies

This section demonstrates how the learned off-policy Lyapunov function (8) can be used to learn stable RL policies. It builds a Lyapunov Soft Actor Critic (LSAC) algorithm by adding the off-policy Lyapunov function to guide the Soft Actor Critic Algorithm [19] to learn the control policy. It also shows that the proposed off-policy Lyapunov function can be applied to on-policy data by building a Lyapunov Proximal Policy Optimization (LPPO) based on the Proximal Policy Optimization Algorithm [20].

Stabilizing Off-Policy Algorithms

The SAC algorithm learns the parameterized policy via maximizing entropy using the loss function:

$$J_\pi(\phi) = \mathbb{E}_{(s,a,r,s')\sim D}\big[\alpha(\log(\pi_\phi(a|s)) + \mathcal{H}) + Q_\theta(s, a)\big], \tag{9}$$

where $\mathcal{H}$ is the minimum entropy and α is the entropy temperature hyperparameter which weighs the relative importance of the entropy.

The proposed LSAC first learns the off-policy Lyapunov function via (8), and then uses it to guide the learning of the control policy through inntroducing the Lie derivative into the SAC policy loss via a Lyapunov temperature hyperparameter β by:

$$J_\pi(\phi) = \mathbb{E}_{(s,a,r,s')\sim D}\big[\alpha(\log(\pi_\phi(a|s)) + \mathcal{H}) + Q(s, a) + \beta \max(0, \mathcal{L}_{f,\Delta t} L_\eta + \mu)\big] \tag{10}$$

If the Lie derivative is negative by the minimum amount μ, then the Lyapunov function does not bias the learning. The agent is only penalized for taking actions that cause the Lie derivative to be positive.

Figure 1 shows the full algorithm.

Extension to On-Policy Algorithms

PPO is an on-policy algorithm that learns a policy that maximizes the advantage $\hat{A}_t$, which measures the difference between the state-action pair and the expected value of the state, using the following loss function:

$$J_\pi(\phi) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\min\left(\frac{\pi_\phi}{\pi_{old}}\hat{A}_t,\ \text{clip}\left(\frac{\pi_\phi}{\pi_{old}}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{11}$$

The hyperparameter ϵ controls the clipping of the ratio of the current policy to the sampled policy to prevent large changes in the policy.

The proposed LPPO learns the Lyapunov function using on-policy data, similarly to POLYC [15], but using the off-policy Lyapunov function with the loss defined in (8). Then, it includes the Lyapunov decreasing condition in an augmented advantage $\hat{A}_\beta$ by:

$$\hat{A}_\beta = \hat{A}_t + \beta \min(0, -(\mathcal{L}_{f,\Delta t}L_\eta + \mu)) \tag{12}$$

and replaces $\hat{A}_t$ with $\hat{A}_\beta$ in (11) in the policy loss function (11):

$$J_\pi(\phi) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\min\left(\frac{\pi_\phi}{\pi_{old}}\hat{A}_\beta,\ \text{clip}\left(\frac{\pi_\phi}{\pi_{old}}, 1-\epsilon, 1+\epsilon\right)\hat{A}_\beta\right)\right] \tag{13}$$

As in LSAC, a negative Lie derivative does not bias the learning and a penalty is applied to the advantage when the Lie derivative is positive.

Figure 1 presents the full algorithm.

Stability Certification Stability certificates can be obtained: (i) from the loss (8), which indicates that the Lyapunov conditions are satisfied and a Lyapunov function is found when it converges to zero; and (ii) from the Almost Lyapunov Conditions [21], which certify stability when a small number of bounded violations exist near the equilibrium. The Pendulum-v1 experiment illustrates each method in Figure 2 (b) and in Figure 3, respectively.

4 Experimental Results

In this section, numerical experiments illustrate: (i) the application of the proposed off-policy Lyapunov SAC algorithm (LSAC) to an inverted pendulum; and (ii) how the off-policy Lyapunov function can be applied to a quadrotor, via LPPO, for which on-policy learning has been shown to be advantageous.

4.1 Inverted Pendulum

The first experiment uses the standard Pendulum-v1 environment from Open AI Gym [22], without any modifications to the environment. Because the motor has insufficient torque to drive the pendulum directly to the upright position from all starting states, a swing up is sometimes necessary. The state is the position of the end of the pendulum, $x = \cos\theta$ and $y = \cos\theta$, and its angular velocity $\dot{\theta}$. The action is the torque τ applied by the motor at the joint.

Figure 2 (a) depicts the training rewards of LSAC, SAC, LAC, POLYC and PPO for the Pendulum-v1 environment, for the first 100,000 training steps. Over the 10 random seeds, LSAC achieves

Lyapunov Soft Actor-Critic (LSAC)

Initialize policy π_ϕ, RL value function and action value function Q_θ, V_ψ, $V_{\bar{\psi}}$, Lyapunov function L_η randomly
Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
while steps $< K$ **do**
  for each environment step **do**
    Sample $a_t \sim \pi_\phi(a_t|s_t)$
    Sample $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    steps $\leftarrow$ steps $+1$
  end for
  for each Lyapunov optimization step **do**
    Sample mini batch from $\mathcal{D}$
    Compute J_{L_η} via (8)
    $\eta \leftarrow \eta - \alpha_\eta \nabla_\eta J_{L_\eta}(\eta)$
  end for
  for each policy optimization step **do**
    Sample mini-batch from $\mathcal{D}$
    $\psi \leftarrow \psi - \alpha_\psi \nabla_\psi J_{V_\psi}$
    $\theta \leftarrow \theta - \alpha_\theta \nabla_\theta J_{Q_\theta}$
    Compute J_{π_ϕ} via (10)
    $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_{\pi_\phi}$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end while

Lyapunov Proximal Policy Optimization (LPPO)

Initialize policy π_ϕ, RL value function V_θ Lyapunov function L_η randomly
Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
while steps $< K$ **do**
  $\mathcal{B} \leftarrow \emptyset$
  for $t = 1$ to N **do**
    Sample $a_t \sim \pi_\phi(a_t|s_t)$
    Sample $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  end for
  Sample mini-batches from $\mathcal{B}$
  Compute J_{L_η} via (8)
  $\eta \leftarrow \eta - \alpha_\eta \nabla_\eta J_{L_\eta}(\eta)$
  for each policy optimization step **do**
    Sample mini-batches from $\mathcal{B}$
    $\delta_t \leftarrow r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$
    $\hat{A}(s_t, a_t) \leftarrow \delta_t + \gamma\delta_{t+1} + \dots$
    Compute $\hat{A}_\beta$ via (12)
    Compute J_{π_ϕ} via (13)
    $\phi \leftarrow \phi + \alpha_\phi \nabla_\phi J_{\pi_\phi}$
    $\theta \leftarrow \theta - \alpha_\theta \nabla_\theta J_{V_\theta}$
  end for
  steps $\leftarrow$ steps $+N$
end while

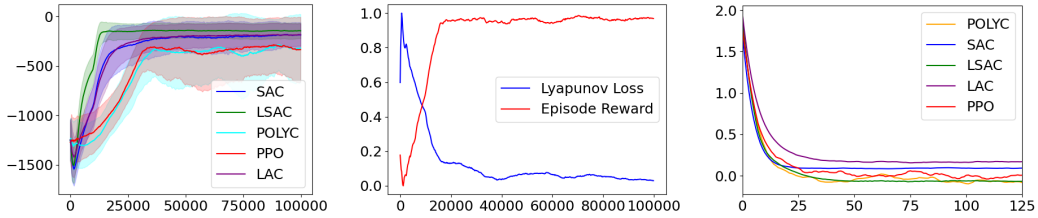
Figure 1: The two proposed algorithms, LSAC (left) and LPPO (right). J_{V_ψ}, J_{Q_θ}, $\bar{\psi}$ are defined in [19], and J_{V_θ} is defined in [20].

the highest reward with the fewest steps to convergence, which indicates that LSAC is the most sample efficient. Figure 2 (b) plots a sample trajectory after all algorithms have been trained. LSAC stabilizes the pendulum closest to the equilibrium $\theta = 0$ with minimal noise. POLYC also stabilizes it near the equilibrium but with more noise, while SAC and LAC stabilize it with minimal noise but further from the equilibrium.

Figure 3 shows, from left to right, the contours of the Lyapunov functions learned by the LSAC, POLYC and LAC. The red dots indicate violations of the Lyapunov decreasing condition along the simulated trajectories. The function learned by LSAC violates the decreasing condition the least, as illustrated by the minimal number of red dots in the left-most plot in Figure 3. The functions learned by POLYC and LAC violate the Lyapunov decreasing condition much more often, as seen in the larger number of red dots in the middle and right-most plots in Figure 3, respectively. If the Almost Lyapunov conditions [21] were to be validated, LASC would have the largest region of attraction.

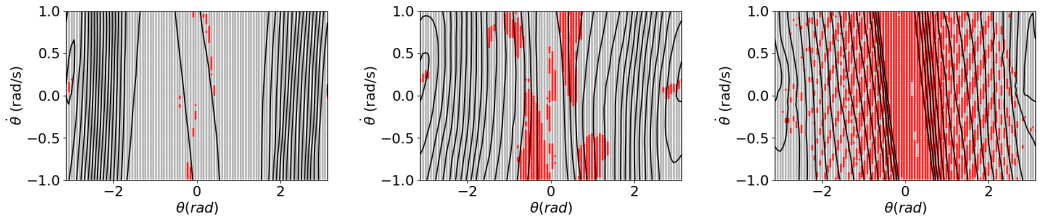
4.2 Quadrotor

Quadrotor control is a difficult problem for model-free RL. As shown in [15], the two off-policy methods SAC and LAC struggle to produce any meaningful controller. Therefore, the numerical experiments in this section integrate the proposed off-policy Lyapunov function into the clean-RL implementation of PPO, which uses a normalized state and reward function for training [23], to learn a trajectory tacking controller for a quadrotor simulated in the Mujoco physics simulator [24]. As in [25], the desired trajectory is generated by providing actions to the quadrotor and recording its state. The quadrotor then learns to track the desired trajectory guided by three algorithms: the proposed LPPO, the POLYC and the PPO algorithms.



(a) Training Rewards (b) LSAC Reward vs Loss (c) Sample Trajectory

Figure 2: Pendulum-v1 Experiment Results: (a) the reward of different algorithms during training, as function of the number of episodes, and with the shaded region showing one standard deviation over the 10 random seeds; (b) the loss (8) and the reward during training (y axis is normalized); (c) a sample trajectory for each algorithm after training is complete.



(a) LSAC - 0.84% Violations (b) POLYC - 13% Violations (c) LAC - 51% Violations

Figure 3: Level curves of the Lyapunov candidates learned by LSAC, POLYC and LAC. Grey dots represent pendulum states where the Lie derivative is negative. Red dots are pendulum states where the Lie derivative is positive.

The implementation extends [26] to track a trajectory. The 13-dimensional quadrotor state comprises the position error ($p_e \in \mathbb{R}^3$), the orientation error represented as a quaternion ($q_e \in \mathbb{R}^4$), the velocity error ($v_e \in \mathbb{R}^3$) and the angular velocity error ($\dot{\theta} \in \mathbb{R}^3$). The 4-dimensional controls are the applied thrust F_z along the z axis of the quadrotor's body frame measured in Newtons, and the angular velocity of the quadrotor along its x, y and z axes measured in rad/s. This choice of controls is justified (i) because motor thrusts map directly to the applied thrust and the body rates, and (ii) because body rates-based controls have better performance than motor thrust-based controls [27].

Since [15] has illustrated that SAC and LAC fail to learn any meaningful quadrotor control policy, this section compares only on-policy algorithms, namely the LPPO, POLYC and PPO algorithms. Figure 4 shows the training rewards. LPPO and POLYC achieve a similar maximum reward while the PPO maximum reward is slightly lower. However, LPPO is more sample efficient as it converges faster than POLYC.

Figure 5 plots sample trajectories after training is complete. LPPO tracks the reference trajectory most accurately. POLYC also tracks the reference trajectory accurately until the very end of the episode. PPO is also able to track the reference trajectory but with larger error compared both to LPPO and to POLYC.

5 Conclusion

This paper has proposed a method for self learning Lyapunov functions on off-policy data. Specifically, it has shown that a Lyapunov function can be effectively learned as the expectation over the actions under the current policy provided it depends both on the state and on the action. The paper has also illustrated how the proposed off-policy Lyapunov function can advise both off policy and on policy RL algorithms. Numerical experiments have demonstrated that the off-policy Lyapunov-based RL algorithms are more sample efficient and can achieve better performance on the Pendulum-v1 and Mujoco Quadrotor environments than existing RL algorithms.

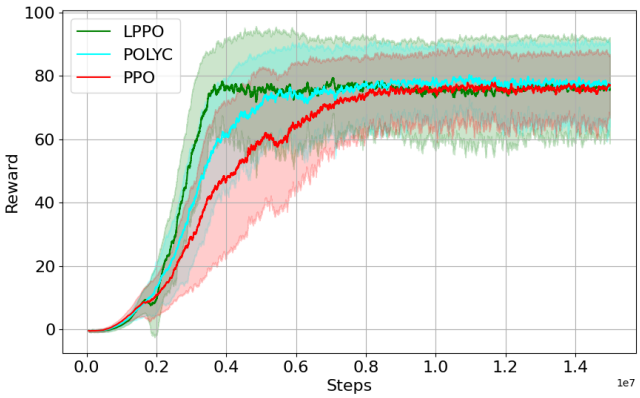
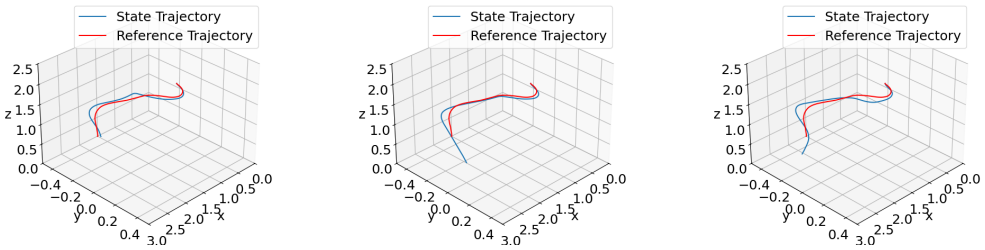


Figure 4: The mean training rewards for LPPO, POLYC, and PPO on the Mujoco Quadrotor environment, obtained from ten random seeds and plotted with a one standard deviation shaded region.



(a) LPPO control. (b) POLYC control. (c) PPO control.

Figure 5: Trajectory tracking for the quadrotor controlled by LPPO, POLYC, and PPO. The drone starts at the same starting point of $(x_0, y_0, z_0) \sim (1, 0, 2)$ for all three algorithms.

6 Limitations

While the experiments in Section 4 show great success in simulated environments, the algorithms presented have yet to be tested in physical environments. A greater number of varied experiments would also aid in verifying the robustness of the proposed algorithms. Testing them in different simulated and physical environments is an important consideration for future work.

The proposed algorithms also include two additional hyperparameters; the minimum rate of decrease μ and the Lyapunov temperature β. The paper provides experimental results after hand tuning these hyperparameters. The inclusion of a hyperparameter sweep and an appropriate discussion is also an important direction for future work.

Because the proposed algorithms build upon an existing algorithm, the success of the underlying algorithm (i.e., SAC or PPO) is necessary for the success of the algorithms in this paper.

This paper proposes a method to learn the Lyapunov function off-policy. Since the Lyapunov function is inherently dependent on the current controller, there is bias in the data collected from previous control policies. The paper proposes a method to address the bias but does not analyze the impact of the bias itself. Further work could compare the proposed method on off-policy and on-policy data, and could further reduce bias through importance sampling.

Lastly, the work presented shows promise in practice, but currently lacks theoretical support. Developing stability guarantees for the proposed algorithms is an important area for future work.

Acknowledgments

The authors thank the reviewers for their constructive comments. They also acknowledge the financial support provided by the National Science and Engineering Research Council of Canada (DG34771)

References

[1] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone. Deep reinforcement learning for robotics: A survey of real-world successes. In *2025 Annual Review of Control, Robotics, and Autonomous Systems - Early Publication*, 2025.

[2] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[3] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(Volume 5, 2022):411–444, 2022. ISSN 2573-5144.

[4] T. Perkins and A. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 01 2002.

[5] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] L. Zhao, K. Gatsis, and A. Papachristodoulou. Stable and safe reinforcement learning via a barrier-lyapunov actor-critic approach. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 1320–1325, 2023.

[7] S. Tonkens and S. Herbert. Refining control barrier functions through hamilton-jacobi reachability. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13355–13362, 2022.

[8] C. Dawson, S. Gao, and C. Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023.

[9] B. Hejase and U. Ozguner. Lyapunov stability regulation of deep reinforcement learning control with application to automated driving. In *2023 American Control Conference (ACC)*, pages 4437–4442, 2023.

[10] R. Cheng, G. Orosz, R. Murray, and J. Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *AAAI Conference on Artificial Intelligence*, volume 22, page 3387–3395, 2019.

[11] P. Osinenko, G. Yaremenko, R. Zashchitin, A. Bolychev, S. Ibrahim, and D. Dobriborsci. Critic as lyapunov function (calf): a model-free, stability-ensuring agent. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 2517–2524, 2024.

[12] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

[14] D. Du, S. Han, N. Qi, H. B. Ammar, J. Wang, and W. Pan. Reinforcement learning for safe robot control using control lyapunov barrier functions. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9442–9448, 2023.

[15] Y.-C. Chang and S. Gao. Stabilizing neural control using self-learned almost lyapunov critics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1803–1809, 2021.

[16] Z. Xiong, J. Eappen, A. H. Qureshi, and S. Jagannathan. Model-free neural lyapunov control for safe robot navigation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5572–5579, 2022.

[17] Y.-C. Chang, N. Roohi, and S. Gao. Neural lyapunov control. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[18] J. Liu, Y. Meng, M. Fitzsimmons, and R. Zhou. Physics-informed neural network lyapunov functions: Pde characterization, learning, and verification. *Automatica*, 175:112193, 2025. ISSN 0005-1098.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Deep Reinforcement Learning Symposium*, 2017.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] S. Liu, D. Liberzon, and V. Zharnitsky. Almost lyapunov functions for nonlinear systems. *Automatica*, 113:108758, 2020. ISSN 0005-1098.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. 06 2016. doi:10.48550/arXiv.1606.01540.

[23] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[24] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[25] D. Sun, S. Jha, and C. Fan. Learning certified control using contraction metric. In *Proceedings of the Conference on Robot Learning*, 2020.

[26] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, Oct 2017. ISSN 2377-3766.

[27] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10504–10510, 2022. doi:10.1109/ICRA46639.2022.9811564.