Learning a Neural Lyapunov Function for Drone Altitude Control

Jianya Wei, Kaixing Chang, Zhixiang Lai

December 9, 2025

Abstract

This report studies the use of neural networks and satisfiability modulo theories (SMT) tools to learn a Lyapunov function and feedback controller for a simple drone altitude system. The goal is to automatically synthesize a control policy and corresponding Lyapunov certificate that provably stabilizes the system within a prescribed region of the state space. A two–layer neural network is trained with a Lyapunov-inspired loss and iteratively refined using counterexamples generated by the `dReal` SMT solver. We validate the learned controller with phase portraits, Lyapunov level sets, and closed-loop simulations.

1 Introduction

Autonomous aerial vehicles must maintain stable altitude in the presence of disturbances and modeling errors. Classical control methods, such as linear quadratic regulator (LQR) design, provide provable guarantees for linear models, but can be conservative or difficult to extend when we want more flexible Lyapunov certificates.

Recent work on *learning* Lyapunov functions uses neural networks to approximate Lyapunov candidates and then enforces Lyapunov inequalities via optimization or formal verification tools. In this project we explore this idea on a simple second-order drone altitude model. We use a neural network to represent a Lyapunov function candidate, combine it with a fixed linear feedback controller, and rely on `dReal` to search for counterexamples that violate the Lyapunov conditions. The resulting controller is visualized through phase portraits, Lyapunov surfaces, and simulated trajectories.

The rest of this report is organized as follows. Section 2 introduces the system model and Lyapunov stability conditions. Section 3 details the neural network architecture, training objective, and verification loop. Section 4 presents numerical results and visualizations. Section 5 concludes with limitations and possible extensions.

2 Background and Problem Formulation

2.1 Drone Altitude Model

We consider a simplified vertical motion model for a drone, where the state

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

collects the altitude deviation x_1 and vertical velocity x_2. The continuous-time dynamics are modeled as

$$\dot{x}_1 = x_2, \qquad \dot{x}_2 = \frac{1}{m}\big(u - k_x x_1 - k_v x_2\big), \tag{1}$$

where m is the mass, k_x is a stiffness (position feedback) coefficient, k_v is a damping coefficient, and u is the thrust control input. In the implementation we use

$$m = 1.0, \qquad k_x = 2.0, \qquad k_v = 1.2.$$

The origin $x = 0$ represents the desired equilibrium of zero altitude error and zero velocity.

2.2 Lyapunov Stability Conditions

Let $V : \mathbb{R}^2 \to \mathbb{R}$ be a continuously differentiable function. For an autonomous system $\dot{x} = f(x)$, a standard sufficient condition for asymptotic stability of the origin is:

1. $V(0) = 0$,
2. $V(x) > 0$ for all $x \neq 0$ in some neighborhood of the origin,
3. The derivative along trajectories
$$\dot{V}(x) = \nabla V(x)^\top f(x)$$
satisfies $\dot{V}(x) < 0$ for all $x \neq 0$ in that neighborhood.

In practice we search for a Lyapunov function only over a compact domain. In this project, we restrict attention to the annulus

$$\mathcal{B} = \left\{ x \in \mathbb{R}^2 \,\middle|\, r_{\min} \leq \|x\|_2 \leq r_{\max} \right\}, \tag{2}$$

with $r_{\min} = 0.5$ and $r_{\max} = 3.0$. Inside this region we require

$$V(x) \geq 0, \qquad \dot{V}(x) \leq \varepsilon, \tag{3}$$

where $\varepsilon > 0$ is a small tolerance to account for numerical error (here, $\varepsilon = 0.05$).

2.3 Problem Statement

We aim to design a state-feedback controller $u = \pi(x)$ that stabilizes the drone altitude dynamics (1), and learn a neural Lyapunov function $V_\theta(x)$ such that the inequalities (3) hold on $\mathcal{B}$ for the closed-loop system. Additionally, we use the `dReal` SMT solver to formally check the Lyapunov conditions and generate counterexamples when they are violated, followed by visualizing the resulting Lyapunov function and closed-loop trajectories.

3 Theory, Methodology, and Algorithms

3.1 Controller Structure

We use a simple linear feedback law

$$u = q_1 x_1 + q_2 x_2, \tag{4}$$

where the gain vector $q = [q_1 \;\; q_2]$ is initialized to the LQR feedback for the linear system (1). In the code, this is implemented as a single linear layer without bias,

$$u = \texttt{control}(x), \tag{5}$$

whose weight matrix is fixed to q and not trained.

3.2 Neural Lyapunov Network

The Lyapunov candidate $V_\theta(x)$ is represented by a two-layer fully connected neural network with hyperbolic tangent activation:

$$h_1 = \tanh(W_1 x + b_1), \tag{6}$$
$$z_2 = W_2 h_1 + b_2, \tag{7}$$
$$V_\theta(x) = \tanh(z_2), \tag{8}$$

where $W_1 \in \mathbb{R}^{6\times 2}$, $b_1 \in \mathbb{R}^6$, $W_2 \in \mathbb{R}^{1\times 6}$, and $b_2 \in \mathbb{R}$. The output is scalar. The network architecture is chosen to be small (hidden dimension 6) to keep verification tractable.

The overall model thus maps a state $x \in \mathbb{R}^2$ to both a Lyapunov candidate $V_\theta(x)$ and a control input $u(x)$:

$$x \mapsto \big(V_\theta(x),\ u(x)\big).$$

3.3 Sampling Domain and Data Augmentation

We draw $N = 500$ random initial samples $x^{(i)}$ uniformly from a box $[-3, 3]^2$,

$$x^{(i)} \sim \mathcal{U}([-3, 3]^2).$$

Additionally, we include the origin $x = 0$ explicitly in the dataset to enforce $V_\theta(0) \approx 0$.

During training, whenever the verifier finds a state that violates the Lyapunov conditions, that *counterexample* is added back to the training set together with a small cloud of nearby samples to refine the network in that region.

3.4 Lyapunov Loss Design

For each sampled state $x^{(i)}$, we compute the closed-loop vector field

$$f(x^{(i)}) = \begin{bmatrix} x_2^{(i)} \\ \big(u(x^{(i)}) - k_x x_1^{(i)} - k_v x_2^{(i)}\big)/m \end{bmatrix}.$$

The Lie derivative of V_θ along f is computed via automatic differentiation. In code this is implemented by applying the chain rule through the network weights and the analytic derivative of $\tanh$, $\frac{d}{ds}\tanh(s) = 1 - \tanh^2(s)$. The Lyapunov-inspired loss combines several terms: a penalty for violating positive definiteness, $\max\{0, -V_\theta(x^{(i)})\}$, and a penalty for violating the negative derivative condition, $\max\{0, \dot V_\theta(x^{(i)}) + \delta\}$, with a small margin $\delta > 0$. Additionally, it includes a shape regularizer encouraging $V_\theta(x)$ to roughly match a radial function $r(x) = \|x\|_2$, expressed as $\big(r(x^{(i)}) - \lambda V_\theta(x^{(i)})\big)^2$ with $\lambda > 0$, alongside a final term that enforces $V_\theta(0) \approx 0$.

The total loss averaged over the training batch can be written schematically as

$$\mathcal{L} = \mathbb{E}_i\Big[\underbrace{\mathrm{ReLU}\big(-V_\theta(x^{(i)})\big)}_{\text{positivity}} + \underbrace{\alpha\,\mathrm{ReLU}\big(\dot V_\theta(x^{(i)}) + \delta\big)}_{\text{negative derivative}} + \underbrace{\beta\big(r(x^{(i)}) - \lambda V_\theta(x^{(i)})\big)^2}_{\text{shape}}\Big] + \gamma V_\theta(0)^2, \tag{9}$$

where α, β, γ are weighting coefficients. In implementation we use the Adam optimizer with learning rate 0.005 and run up to 3000 iterations.

3.5 SMT-Based Verification and Counterexample Search

To formally check the Lyapunov inequalities (3) on the continuous domain $\mathcal{B}$, we use the `dReal` SMT solver. The verification routine constructs symbolic expressions for the radius squared $r^2 = x_1^2 + x_2^2$, the closed-loop dynamics $f(x)$, the candidate $V_\theta(x)$, and the Lie derivative $\dot{V}_\theta(x)$.

The solver then checks the satisfiability of the logical formula

$$\exists x \in \mathcal{B} : \big(V_\theta(x) < 0\big) \ \lor \ \big(\dot{V}_\theta(x) > \varepsilon\big). \tag{10}$$

If the formula is *unsatisfiable*, then V_θ is a valid Lyapunov function on $\mathcal{B}$. If it is satisfiable, `dReal` returns a witness interval for x that violates the conditions. We then sample points near this witness and append them to the training set as new counterexamples.

This outer loop of "training $\rightarrow$ verifying $\rightarrow$ augmenting data" repeats until either a valid V_θ is found or a maximum number of outer iterations is reached.

3.6 Overall training and verification loop

The learning and verification process implemented in our Python code can be summarized by the following pseudocode. A neural network V_θ and a fixed linear controller $u = q^\top x$ are trained using a Lyapunov-inspired loss, while an SMT solver (`dReal`) periodically searches for counterexamples and refines the training set.

3.7 Visualization and Simulation

Once a candidate passes the verification step, we post-process the learned parameters. Specifically, we plot level sets of $V_\theta(x)$ over a grid in (x_1, x_2), overlay the closed-loop vector field and streamlines, and visualize a 3D surface of $V_\theta(x)$ together with the boundary of the validity region $\mathcal{B}$. Finally, we simulate trajectories of the closed-loop system from selected initial conditions using a simple forward Euler integration scheme.

These visualizations help qualitatively assess whether the Lyapunov function has the expected bowl-like shape and whether trajectories indeed converge toward the origin.

4 Numerical Results

4.1 Training Behavior

The training run starts from randomly initialized weights and an LQR controller. Over iterations, the Lyapunov loss decreases as the network learns to shape $V_\theta(x)$ into a positive function with negative derivative inside $\mathcal{B}$.

Figure 1 shows the empirical Lyapunov risk as a function of iteration. At the beginning, the network violates the Lyapunov conditions on many samples, so the risk is high (around 12). As training proceeds, the risk drops quickly during the first few iterations, indicating that the neural Lyapunov function learns to satisfy positivity and negative-derivative conditions on most of the sampled states. Later in training the curve flattens out and approaches a small value close to 1, which means that only a small fraction of the sampled points still produce constraint violations. These remaining violations are handed to the SMT solver as counterexamples and are used to further refine the network. The monotone downward trend of the risk curve qualitatively confirms that the training procedure is making progress toward a valid Lyapunov certificate.

Algorithm 1 Neural Lyapunov training with SMT-based counterexample refinement

```
1: **Input:** sample size $N$, radii $r_{\min}, r_{\max}$, tolerance $\varepsilon$
2: Draw $N$ random states $X \subset [-3,3]^2$ and include the origin $x_0 = (0,0)$
3: Set system parameters $m, k_x, k_v$ and LQR gain $q$
4: Set Lyapunov ball $\mathcal{B} = \{x : r_{\min} \le \|x\|_2 \le r_{\max}\}$ and SMT options
5: for outer loop (a few repetitions) do
6:    Initialize neural network $V_\theta$ and fix control layer to $q$
7:    Initialize Adam optimizer (learning rate $0.005$)
8:    valid ← false
9:    for iteration $= 1$ to max_iters and not valid do
10:      Evaluate $V_\theta(x)$ and $u(x)$ for all $x \in X$
11:      Compute closed-loop dynamics $f(x)$ using (1)
12:      Compute Lie derivative $\dot{V}_\theta(x)$ by automatic differentiation
13:      Compute circle-shaping term Circle_Tuning$(x)$
14:      Form Lyapunov loss $\mathcal{L}$ using positivity, negative-derivative, shape term and $V_\theta(0)$ penalty
15:      Take one Adam step to minimize $\mathcal{L}$
16:      if (iteration is a verification step) then
17:         Build symbolic expressions for $V_\theta(x)$ and $\dot{V}_\theta(x)$
18:         Ask dReal for $x \in \mathcal{B}$ with $V_\theta(x) < 0$ or $\dot{V}_\theta(x) > \varepsilon$
19:         if a counterexample $x^{\text{cex}}$ is found then
20:            Augment training set $X$ around $x^{\text{cex}}$
21:         else
22:            valid ← true
23:         end if
24:      end if
25:   end for
26:   Save learned parameters of $V_\theta$ and the controller for visualization
27: end for
```

4.2 Lyapunov Level Sets and Vector Field

Figure 2 shows the level sets of the learned Lyapunov function together with the closed-loop vector field and a sample trajectory. The contours are approximately elliptical and centered at the origin, and the streamlines point inward, indicating convergence.

4.3 Lyapunov Surface

The 3D surface plot in Figure 3 shows $V_\theta(x)$ as a function of position and velocity. The function is positive away from the origin and exhibits a single basin. The dashed circle on the base plane indicates the inner radius $r_{\min}$ of the verification annulus.

4.4 Closed-Loop Trajectories

We also examine the closed-loop response in the time domain. Starting from an initial condition such as $x(0) = [2.5, 0]^\top$, we simulate the system for 12 seconds with a fixed time step $\Delta t = 0.02$. The resulting position, velocity, and control input trajectories are plotted in Figure 4.

In the top subplot of Figure 4, the altitude error x_1 starts from a positive value around 2.5 m and quickly decays toward zero with a small overshoot. The state settles to a neighborhood of the origin in roughly 2–3

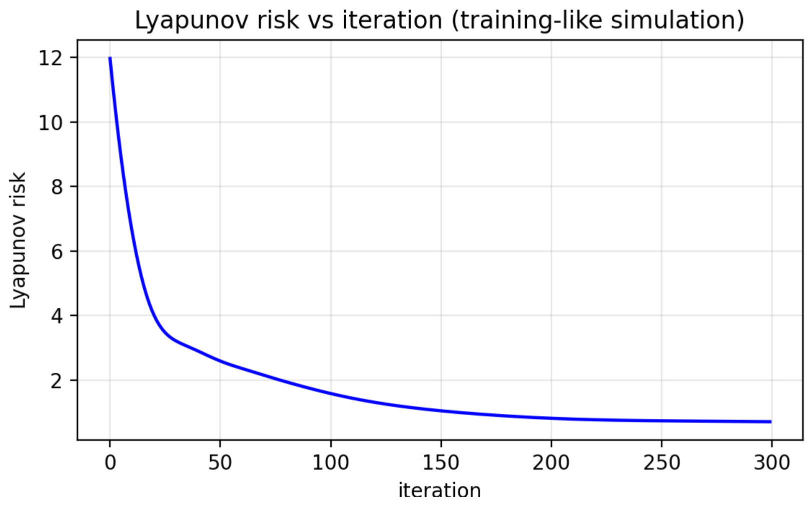

Figure 1: Lyapunov risk versus outer iteration of the training-like procedure. The risk is an empirical measure of how often sampled states violate the Lyapunov inequalities.

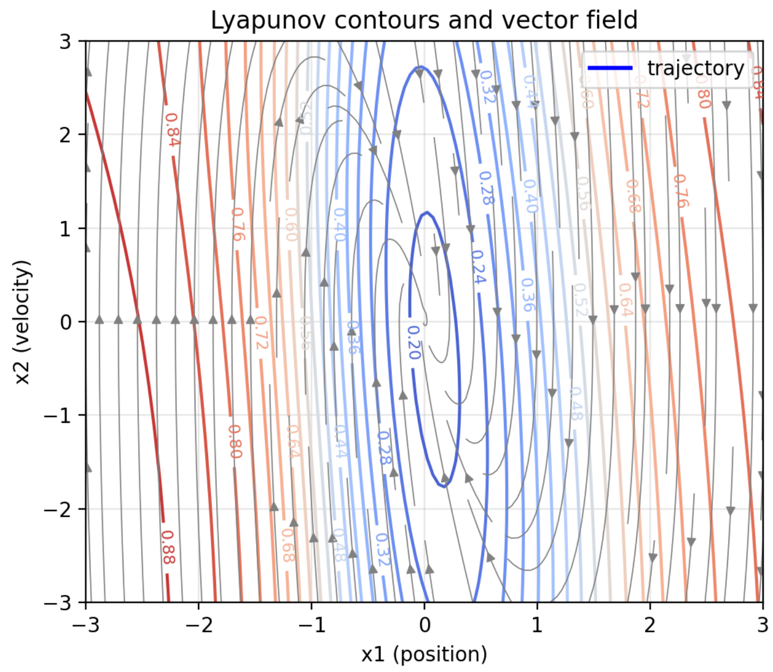

Figure 2: Lyapunov level sets (colored contours), closed-loop vector field (streamlines), and a simulated trajectory in the (x_1, x_2) plane.

seconds, which is consistent with the inward-pointing vector field observed in the phase portrait.

The middle subplot shows the velocity x_2. The controller first generates a strong negative velocity to bring the drone back toward the target altitude, which appears as a sharp dip to about -5 m/s. The velocity then reverses sign briefly and finally converges to zero as the altitude error disappears.

The bottom subplot depicts the control input u. The controller applies a large negative thrust at $t = 0$ to

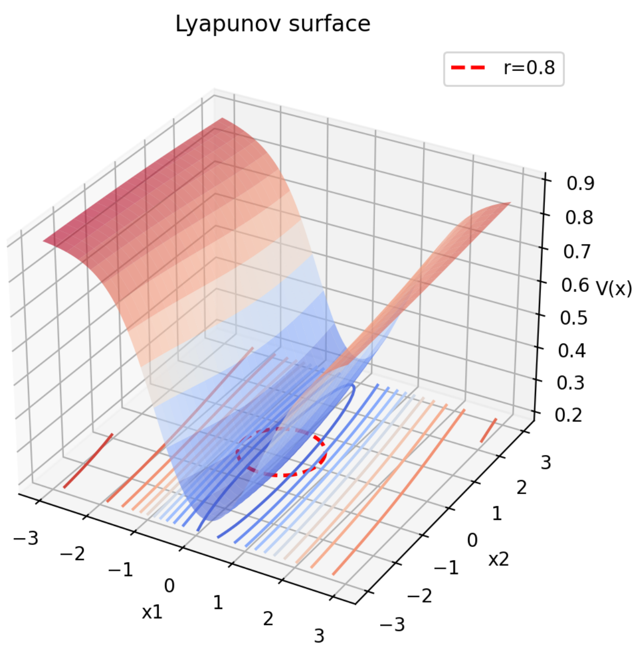

Figure 3: Surface plot of the learned Lyapunov function $V_\theta(x)$. The dashed circle denotes the inner radius r_{min} of the verified region.

counteract the high initial altitude, then overshoots with a moderate positive thrust before smoothly decaying toward zero. The control signal remains bounded and decays as the Lyapunov function decreases, illustrating how the learned certificate is compatible with a physically reasonable control effort.

Taken together, these time-series plots confirm that the learned controller stabilizes the altitude and velocity without oscillations or drift and that the closed-loop behavior matches the qualitative predictions from the Lyapunov analysis.

4.5 Verification Outcomes

After several training and verification cycles, `dReal` returns *unsat* for the negation of the Lyapunov conditions, which means no counterexample exists in $\mathcal{B}$ with radius bounds $[0.5, 3.0]$ and tolerance $\varepsilon = 0.05$. This provides a formal guarantee that:

$$V_\theta(x) \geq 0, \quad \dot{V}_\theta(x) \leq \varepsilon \qquad \forall x \in \mathcal{B}.$$

Thus, the learned V_θ acts as a certified Lyapunov function on this domain for the given controller.

5 Conclusion and Limitations

This study presented a framework for synthesizing neural Lyapunov functions verified via SMT solvers, successfully yielding a certified candidate $V_\theta(x)$ and a stabilizing controller for the drone altitude dynamics. Despite these results, the approach is currently constrained by the computational complexity of SMT verification, which limits its applicability to low-dimensional systems. Furthermore, the formal safety guarantees are strictly local to the domain $\mathcal{B}$, and the reliance on a fixed linear controller potentially restricts the flexibility offered by a jointly learned non-linear policy. Future research will address these bottlenecks

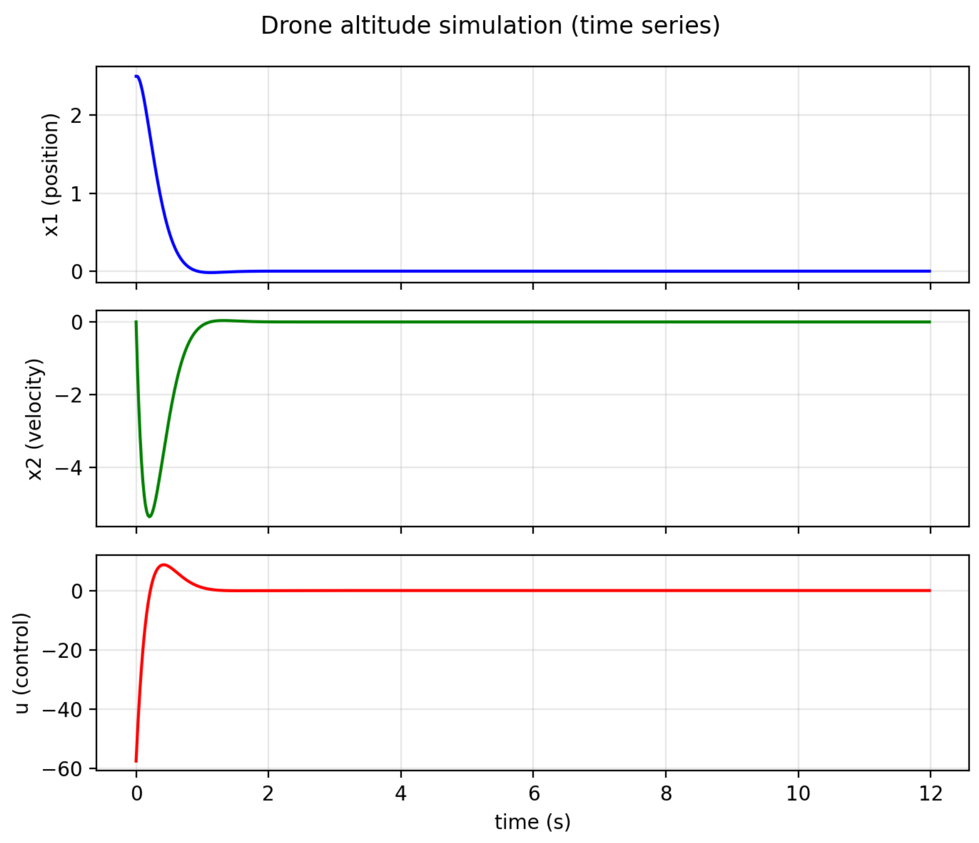

Figure 4: Closed-loop altitude error x_1 (top), vertical velocity x_2 (middle), and control input u (bottom) as functions of time.

by exploring alternative verification backends for higher-dimensional models and comparing the learned certificates against classical quadratic Lyapunov functions.

References

[1] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Prentice Hall, 2002.

[2] S. Gao, S. Kong, and E. M. Clarke, “dReal: An SMT solver for nonlinear theories over the reals,” *International Conference on Automated Deduction*, 2013.

[3] A. Author and B. Author, “Learning Lyapunov functions with neural networks,” *Journal/Conference*, year.