SAPIENZA
UNIVERSITÀ DI ROMA

Master's Degree in Control Engineering

Neural Lyapunov Control

Intelligent & Hybrid Control

Professor:
Alessandro Giuseppi

Students:
Coletta Emanuele
2001600

Academic Year 2024/2025

Contents

1 Introduction

In control theory, stability analysis of nonlinear systems is done mainly through Lyapunov functions (LFs), which offer a sufficient condition to prove the stability (simple or asymptotic) of the system. Those functions are however, hard to find in general and require onerous manual work [1]. Furthermore, being only sufficient conditions, the inability to find a Lyapunov function does not entail instability of the equilibrium of interest, and thus does not provide any extra information.

Recently, efforts have been made to learn Lyapunov functions using neural networks (NNs). This report mainly follows on the work of [2]: their framework uses a NN to find a controller and a Lyapunov function for the closed loop system, plus a *falsifier* to ensure the NN actually satisfies the conditions for the function to be Lyapunov. The neural network minimizes the *Lyapunov Risk* (Section 2) and uses *tanh* activation functions to ensure that the resulting LF is smooth. The falsification step is formulated as a δ-complete constraint satisfaction problem [3], ensuring that if no violation is found in the domain of interest, the Lyapunov conditions are guaranteed to be satisfied.

This work is organized as follows: in section 2, the framework and the theory behind it are explained; in section 3 experiments are carried out on different systems. Finally, conclusions are drawn in section 4.

2 Theory

2.1 Lyapunov stability

Consider an autonomous system

$$\dot{x} = f(x) \tag{1}$$

with $f : D \subset \mathbb{R}^n \to \mathbb{R}^n$ locally Lipschitz, and with the origin $x = 0$ an equilibrium, i.e. such that $f(0) = 0$.

Definition 2.1. The equilibrium $x = 0$ for (1) is:

- stable: if $\forall \epsilon > 0, \exists \delta > 0$ such that

$$||x(0)|| < \delta \implies ||x(t)|| < \epsilon, \forall t \geq 0$$

- asymptotically stable: if it is stable and δ is such that

$$||x(0)|| < \delta \implies \lim_{t \to \infty} x(t) = 0$$

- unstable: if it is not stable

Theorem 1. Let $x = 0$ be an equilibrium for (1) and $D \subset \mathbb{R}^n$ a subset of the domain containing $x = 0$. Let $V : D \to \mathbb{R}$ a continously differentiable function such that

$$V(0) = 0 \text{ and } V(x) > 0 \text{ in } D \setminus \{0\}$$

$$\dot{V}(x) \leq 0 \text{ in } D$$

then, x=0 is stable.

Furthermore, if

$$\dot{V}(x) < 0 \text{ in } D$$

$x = 0$ is asymptotically stable

where $\dot{V}(x) = L_f V(x) = \nabla_x^T V(x) f(x)$ is the *Lie derivative along f of $V(x)$*. Note also that the form (1) includes the case of general affine nonlinear systems of the form

$$\dot{x} = f(x) + g(x)u \tag{2}$$

under state feedback $u = \alpha(x)$.

2.2 Neural network structure

The neural network is a multilayer feedforward neural network. Notably, ReLU functions cannot be used since the falsification step needs to check conditions on the Lie derivative as well, which must exist. Non-smooth functions like the ReLU do not ensure the existence of the lie derivative, thus tanh or sigmoid functions should be used. Throughout this work, the tanh function is used as activation function.

The neural network is actually composed of two networks. The first computes a control, in the form $u = Kx$, the second computes a Lyapunov function for the closed loop system. Since stochastic gradient descent can get stuck in local minima (especially for small networks like the one at hand), the weights for the control are initialized to the LQR solution for the system, if any.

The loss function is built so to estimate the "degree of violation" of the Lyapunov conditions for asymptotic stability, which are:

- $V(x)$ is positive definite
- $L_f V(x)$ is negative definite
- both functions are zero at the origin

Note with θ and u the neural network parameters for the Lyapunov function and control outputs, respectively. Denote with $V_\theta(x)$ the Lyapunov function encoded by the NN, which depends on the parameters θ of the network itself; with f_u the closed-loop system depending on the control u learnt by the neural network and denote with $L_{f_u} V_\theta(x)$ the Lie derivative of $V_\theta(x)$ w.r.t f_u. Then, the control design problem can be written as a minimization of the minimax function:

$$\inf_{\theta,u} \sup_{x \in D} \Big(\max(0, -V_\theta(x)) + \max(0, L_{f_u} V_\theta(x)) + V_\theta^2(0) \Big) \tag{3}$$

To guarantee stability, these conditions need to be satisfied over all states in D. The inner maximization is taken care of by the falsifier (section 2.3), while for the learning step, we define the following:

Definition 2.2. (Lyapunov Risk) Consider a candidate Lyapunov function V_θ for the closed loop system of the form (1). The Lyapunov risk is defined as

$$L_\rho(\theta, u) = \mathbb{E}_{x \sim \rho(\mathcal{D})} \Big(\max(0, -V_\theta(x)) + \max(0, L_{f_u} V_\theta(x)) + V_\theta^2(0) \Big), \tag{4}$$

where x is a random variable over the state space of the system, with distribution ρ. In practice, we work with the Monte Carlo estimate of , that is the *empirical Lyapunov risk*, defined as

$$L_\rho(\theta, u) = \frac{1}{N} \sum_{i=1}^{N} \Big(\max(0, -V_\theta(x_i)) + \max(0, L_{f_u} V_\theta(x_i)) \Big) + V_\theta^2(0), \tag{5}$$

where x_i are the samples of the state vectors according to $\rho(\mathcal{D})$.

The empirical Lyapunov risk is an unbiased estimator of the Lyapunov risk function. Also, L_ρ is positive semidefinite, and any (θ, u) that corresponds to a true Lyapunov function satisfies $L(\theta, u) = 0$, which means that Lyapunov functions define global minimizers of the Lyapunov risk function.

2.3 Falsifier

In the falsification step, the Lyapunov falsification is written as the following first-order logic formula over the reals.

$$\Phi_\epsilon(x) := \Big(\sum_{i=1}^{n} x_i^2 \geq \epsilon \Big) \wedge \Big(V(x) \leq 0 \vee \nabla_{f_u} V(x) \geq 0 \Big), \tag{6}$$

where $\epsilon \in \mathbb{Q}^+$ is a small positive constant which bounds the tolerable numerical error and where x is bounded in the state space D of the system. The constant ϵ is specifically chosen to control numerical sensitivity near the origin, and in particular to avoid problems like arithmetic underflow. Its value should be chosen to be orders of magnitude smaller than the range of the state variables, i.e. $\sqrt{\epsilon} \ll \min(1, ||D||_2)$.

The solution of the falsification constraint $\Phi_\epsilon(x)$ requires global minimization of highly nonconvex functions (since the Lie derivatives are involved), which is a NP-hard task. In this work, falsification constraints are solved using the dReal [4] SMT (*Satisfiability Modulo Theory*) solver. To certify the Lyapunov conditions, the falsifier must always report solutions if there are any. In this sense, dReal satisfies the δ-completeness property [3].

This ultimately means that, chosen a constant $\delta \in \mathbb{Q}^+, \delta \ll \epsilon$ if the falsifier returns no solution for the δ-relaxed problem (i.e. it is not satisfiable), no solution exists for the original problem. If no states violating the constraint are found, the candidate is a valid Lyapunov function. On the other hand, if some states are found such that the Lyapunov constraint is violated, they are added to the dataset as counter example and the training of the network continues.

2.4 Tuning the region of attraction

With the loss function (4) an extra term can be added so to try increase the region of attraction of the equilibrium by adding a term of the form $-\alpha V_\theta(x_i)$ to the loss function (5), where α a tunable hyperparameter.

2.5 Algorithm

The resulting algorithm for training formulated by [2] is the following:

Algorithm: Neural Lyapunov Control

function *LEARNING(X,f,u_{lqr})*:

α learning rate

D_{in} input dimension (# state variables of X)

output dimension is 1

$u \leftarrow u_{lqr}$

$V_\theta(x), u(x) \leftarrow \mathrm{NN}_{\theta,u}(x)$

$L_{f_u} V_\theta(x) \leftarrow \sum_{i=1}^{D_{in}} \frac{\partial V(x)}{\partial x_i} f_{u,i}(x)$

Compute Lyapunov risk $L(\theta, u)$

$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta, u)$

$u \leftarrow u + \alpha \nabla_u L(\theta, u)$

return V_θ, u;

endfunction

function *FALSIFICATION(f,u,$V_\theta, \epsilon, \delta$)*:

Encode conditions 6

Run SMT solver with ϵ, δ to verify conditions

return satisfiability, counterexamples

endfunction

function *MAIN(f,$u_{lqr}, \epsilon, \delta, D$)*:

$X \leftarrow D$

while *satisfiable* **do**

$V_{\theta,u}(x) \leftarrow X$ LEARNING(X, f, u_{lqr}) satisfiable, CE $\leftarrow$ FALSIFICATION$(f, u, V_\theta, \epsilon, \delta$ $X \leftarrow X \cup CE$

end

endfunction

3 Experiments

Four experiments have been carried out: the first considers a simple integrator, to prove the approach works for the simplest linear system; then, three nonlinear systems are tested: an inverted pendulum, a cart-pole, and the pendubot.

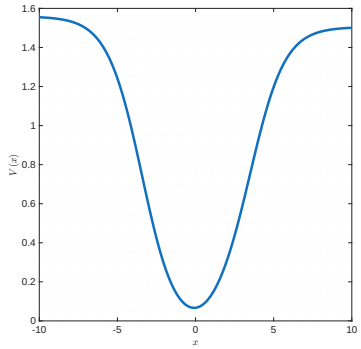
For all the examples where the LQR solution can be computed, the values of Q and R are set to the identity matrix of the appropriate dimension and the controller is computed by solution of the algebraic Riccati equation, using the function `lqr()` in MATLAB. When possible, Lyapunov functions and regions of attractions (ROAs) are plotted and discussed. All experiments also include a simulation of the system comparing LQR and the controller found by the Neural Network.

All tests are run on a desktop computer with an AMD Ryzen 7 3700X CPU, 64GB of DDR4 RAM and a NVIDIA RTX3060 GPU with 12GB of VRAM. Neural network training is always run on the GPU, while falsification is always run on the CPU. Since falsification is the step that takes the most time for more complex systems like the cart-pole and the pendubot, falsification is run in parallel on 12 of the CPU's 16 threads, which sensibly reduces computation times.

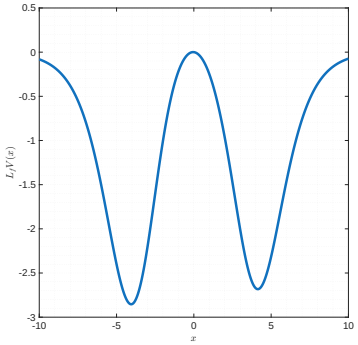
For all the tests, the neural network has a simple multilayer feedforward structure, with the input layer having as many nodes as components of the system in question, one hidden linear layer (6 nodes for integrator and inverted pendulum, 10 for cart-pole and pendubot) with *tanh* activation function as one linear output node with no activation function. A falsification is performed every 20 iterations of NN training, and counter examples added to the training dataset. To avoid getting stuck in a local minima, every 10000 iterations the network is initialized again with random weights, but the dataset of found counterexamples is kept. A test is considered failed if no solution is found after 5 re-initializations and the loss value being approximately zero.

The output node differs from the structure used in the code [5] released by the authors of [2], but is more in line with conventional NN implementations. Other improvements done in this work include a more extensive use of PyTorch's tensor type to optimize computations, and corrections to a computation error in the Lie derivative for the loss function committed in [5]. The latter in particular has been completely replaced by PyTorch's autograd functionality and required the removal of the activation function from the output node. Before doing so, the loss would quickly approach zero but the falsifier would always find a counter example.

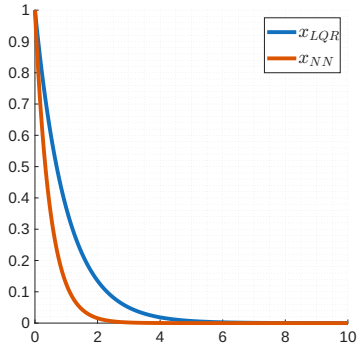
Finally, since the falsification is a NP-hard problem, choosing a valid region that is too large might make the problem practically unsolvable. Relaxing the conditions by setting appropriate values for ϵ and δ may help in finding a solution faster. Great care should still be used in respecting the condition $\delta \ll \epsilon$, as failure to do so might result in a function that is deemed valid by the falsifier but actually leads to a controller that does not stabilize the system.



(a) Lyapunov function learnt by the NN.



(b) Lie derivative.



(c) Comparison of the two controllers.

Figure 1: Integrator.

Table 1 compares execution times for the four tests.

3.1 Integrator

The first test is a simple integrator

$$\dot{x} = u \tag{7}$$

The system being so simple allows for the valid region to be quite large, while still taking reasonable time to solve the falsification problem.

The LQR solution is simply $u = -x$, while the NN finds a control $u =$. The learned Lyapunov function and its Lie derivative, as well as the comparison of the two controllers, is shown in Figure 1.

The Lyapunov function learn by the network is

$$\begin{aligned} V(x) = {} & 0.726 \tanh(0.00107x + 1.55) - 0.385 \tanh(0.51x + 1.56) \\ & - 0.129 \tanh(0.282x + 0.307) + 0.236 \tanh(0.404x - 0.761) \\ & + 0.613 \tanh(0.466x - 1.72) - 0.365 \tanh(0.465x + 1.84) + 0.868 \end{aligned}$$

3.2 Inverted Pendulum

The state variables are the angular position $x_1 = \theta$ and velocity $x_2 = \dot{\theta}$. The state space model is:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{ml^2}(mgl \sin\theta - bx_2 + u) \end{aligned} \tag{8}$$

with $m = 0.1$ kg, $l = 0.5$ m, $b = 0.1$ kg/ms.

To compute the LQR solution, the system is linearized around the equilibrium $x = 0$:

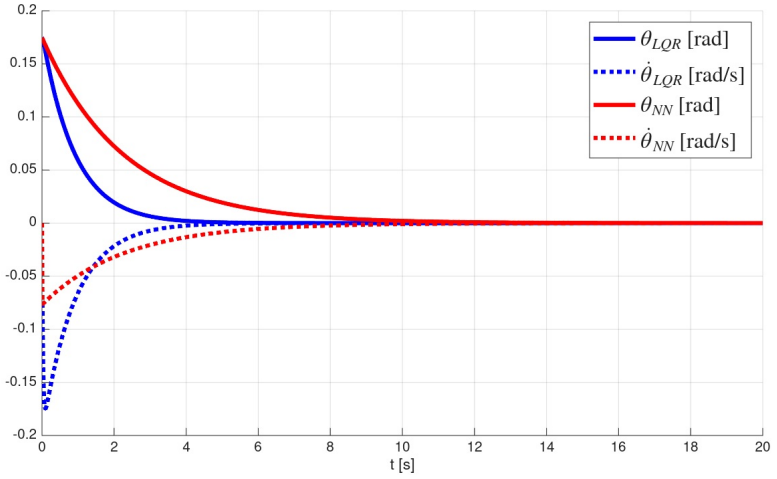


Figure 2: Evolution of the inverted pendulum system under the two controllers

$$A = \begin{bmatrix} 0 & 1.0000 \\ 19.6200 & -2.6667 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 26.6667 \end{bmatrix}$$

The solution to the algebraic Riccati equation yields a control $u = -2.3299x_1 - 1.3781x_2$. The neural network instead learns a control of the form $u = -7.226736x_1 - 14.6675415x_2$.

The equilibria for the closed loop system are all the states such that

$$\begin{cases} \dot{x}_2 = 0 \\ mgl \sin x_1 + k_1 x_1 = 0, \end{cases}$$

which, for this particular choice of m, l and k_1, make the origin the only equilibrium of the closed loop system with both controllers.

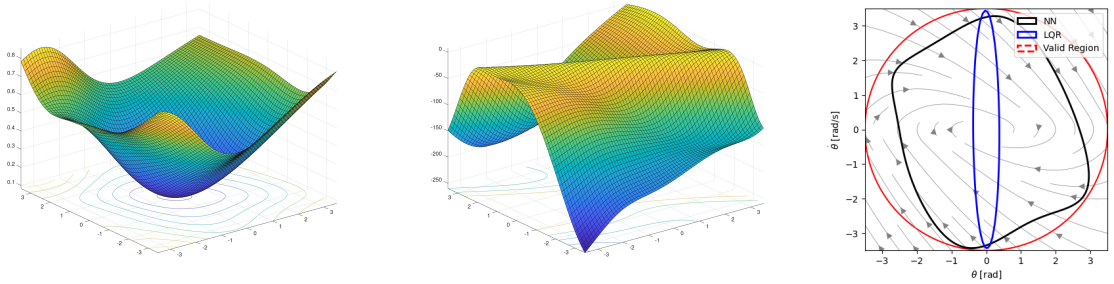
The Lyapunov function learnt by the neural network is

$$\begin{aligned} V(\theta, \dot{\theta}) = {} & 0.184 \tanh(0.749\theta + 1.06\dot{\theta} - 0.122) - 0.154 \tanh(1.1\theta + 1.48\dot{\theta} + 0.533) \\ & - 0.166 \tanh(0.606\theta - 0.37\dot{\theta} + 1.17) - 0.514 \tanh(0.196\theta + 0.286\dot{\theta} + 0.689) \\ & - 0.407 \tanh(0.0164\theta - 0.368\dot{\theta} + 0.98) - 0.377 \tanh(0.0719\dot{\theta} - 0.596\theta + 1.1) + 1.23 \end{aligned}$$

3.2.1 Region of attraction

It's natural to say that if the origin is locally asymptotically stable and it is the only equilibrium of the system, then the origin must be globally asymptotically stable. It's still interesting to see how the shape of the region of attraction for the equilibrium changes when comparing LQR and NN. Regions of attraction can be estimated by looking at the level sets of the Lyapunov function, and checking when those remain closed sets, although these are in general conservative estimates [1]. The plots in figure

3c are obtained by manually selecting the largest level set inside the valid region for each Lyapunov function.



(a) Lyapunov function learnt by the NN.

(b) Lie derivative.

(c) ROA comparison with phase plot.

Figure 3: Inverted pendulum

3.3 Cart-pole

The equations for the cart-pole are taken from [6], which extends the work of [2] to discrete time systems, though using a slightly different methodology. A change of coordinate is performed such that the angle is $\theta = 0$ when the pole points upwards.

$$\begin{aligned} \ddot{x} &= \frac{u + m_p \sin\theta(l\dot{\theta}^2 + g\cos\theta)}{m_c + m_p \sin^2\theta} \\ \ddot{\theta} &= -\frac{u\cos\theta + m_p l\dot{\theta}^2 \cos\theta \sin\theta + (m_c + m_p)g\sin\theta}{l(m_c + m_p \sin^2\theta)} \end{aligned} \tag{9}$$

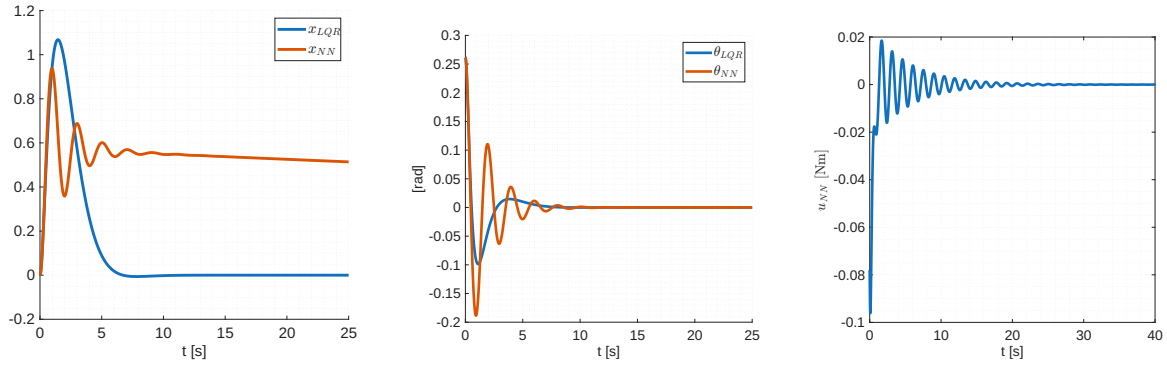
with $m_c = 1.0kg$ the mass of the cart, $m_p = 0.1kg$ the mass of the pole and $l = 1m$ the length of the pole. The state is $q = \begin{bmatrix} x & \theta & \dot{x} & \dot{\theta} \end{bmatrix}^T$, with x and θ the position of the cart and the angle w.r.t the vertical axis.

By linearizing around the origin in the new coordinates, one obtains

$$A = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & -0.9810 & 0 & 0 \\ 0 & 10.7910 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \tag{10}$$

with a LQR solution $u = x + 34.3894\theta + 2.4106\dot{x} + 10.7039\dot{\theta}$.

In this case, the NN failed to learn a Lyapunov function within the limits imposed for training. Notably, the same result is found by [6]. The control found by the NN before training stopped is $u = 0.02846x + 27.7164\theta + 6.3134\dot{x} + 13.7256\dot{\theta}$, and comparison with LQR is shown in Figure 4. Compared with LQR, the intermediate solution found by the NN has a much slower convergence of the x coordinate, which takes about 900s to reach the origin.



(a) (b) (c)

Figure 4: Cart-pole system evolution.

3.4 Pendubot

The pendubot is an underactuated double pendulum, with actuation only on the first link. The dynamic model for the pendubot can be obtained starting from the dynamic model of a 2R planar robot in cartesian space:

$$M(q)\ddot{q} + c(q,\dot{q}) + g(q) = u, \tag{11}$$

with

$$M(q) = \begin{bmatrix} a1 + 2a_2c_2 & a3 + a_2c_2 \\ a3 + a_2c_2 & a3 \end{bmatrix}, c(q,\dot{q}) = \begin{bmatrix} -a_2s_2(\dot{q_2}^2 + 2\dot{q_1}\dot{q_2}) \\ a_2s_2\dot{q_1}^2 \end{bmatrix}, g(q) = \begin{bmatrix} a_4c_1 + a_5c_{12} \\ a_5c_{12}, \end{bmatrix}$$

where a_i are the dynamic parameters defined as in [7], and the shorthand notation $s_i = \sin(q_i), c_i = \cos(q_i), s_{ij} = \sin(q_i + q_j), c_{ij} = \cos(q_i + q_j)$ is used.

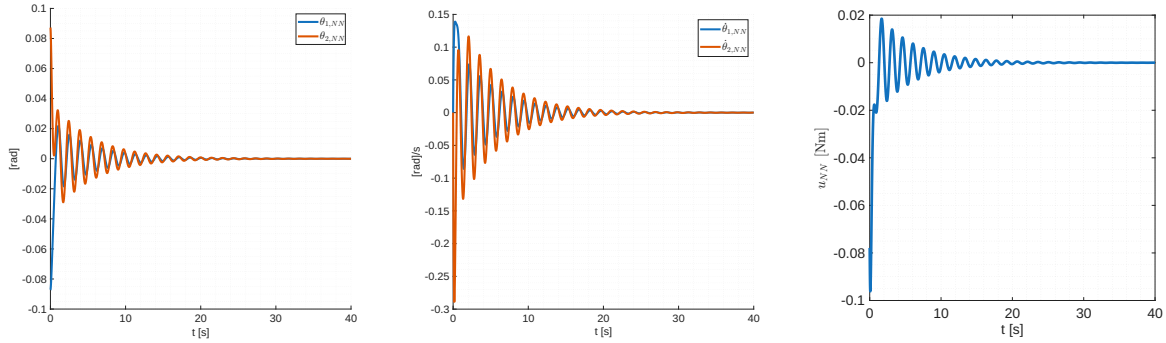
With the change of coordinates $\theta_1' = \theta_1 - \frac{\pi}{2}$, one moves the origin to the configuration where both links point upwards. This only changes the gravity terms $g(q)$ to be:

$$g(q) = \begin{bmatrix} a_4s_1 + a_5s_{12} \\ a_5s_{12} \end{bmatrix}.$$

Since the pendubot is not controllable in the considered configuration, the weights for the control are all initialized to zero. In this case too the NN failed to find a solution within the imposed time limits. The effect of the last control computed by the NN is shown in Figure 5.

3.5 Comparison of the four tests

Table 1 shows a comparison of the training procedure between the four tests. As mentioned before, NN training always makes use of the GPU, while falsification always runs on the CPU using 12/16 threads available. The table is meant to only give a qualitative understanding of the order of magnitudes of the times required to compute



(a) (b) (c)

Figure 5: Pendubot system evolution.

a solution of each system, and different executions will have different results due to the nondeterminism in the random initialization in both PyTorch and dReal. For the cart-pole and pendubot, the framework could not find a valid solution within the training limits. Both tests ended after 5 re-initializations, each with 10000 steps of NN training and falsification every 20 and a loss function value of around 2.5e-4. While in both failed tests no Lyapunov function could be found to verify it, the control computed at the end of training managed the stabilize the respective systems, as shown in Figures 4 and 5.

Test	ϵ	δ	Valid region UB	Training iterations	Training time [s]	Verification time [s]	Valid?
Integrator	0.1	0.01	10	220	15.25	0.02	Yes
Inverted pendulum	0.5	0.01	3.5	2560	18.48	8.08	Yes
Cart-pole	0.02	0.001	0.1	50000	900	350	No
Pendubot	0.02	0.001	0.05	50000	790	95	No

Table 1: Tests comparison

4 Conclusions

This report follows on the work of [2], which introduces a framework for learning a control and valid Lyapunov functions for the closed loop system via neural networks, ensuring Lyapunov stability conditions are verified with a SMT solver. The framework limits itself to controls that are linear in the state of the system, and thus limited to a given region of attraction, which can however be maximized with appropriate terms in the loss function for the training of the neural network. This report tests the approach on four different systems: one linear and three nonlinear systems, two of which underactuated. While the framework performs well on simple systems, it fails to find valid control policies for more complex systems like the cart-pole and the pendubot. Future work may consider larger networks and nonlinear controls, but the main limitation of the method remains the falsification step, which is a NP-hard problem that becomes increasingly difficult to solve in the dimension of the valid region, which is also a hyperparameter that requires tuning to obtain a valid solution.

References

[1] Hassan K. Khalil. *Nonlinear systems*. Prentice Hall, 2002.

[2] Ya-Chien Chang, Nima Roohi, and Sicun 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.

[3] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2012.

[4] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer, 2013.

[5] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control github repository. visited 14th December 2025.

[6] Junlin Wu, Andrew Clark, Yiannis Kantaros, and Yevgeniy Vorobeychik. Neural lyapunov control for discrete-time systems, 2023.

[7] Alessandro De Luca. *Robotics 2 Course Slides*, 2025. Available at `https://www.diag.uniroma1.it/deluca/rob2_en/material_rob2_en.html`, visited 14th December 2025.