IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 June 2023, accepted 23 June 2023, date of publication 3 July 2023, date of current version 11 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3291349

APPLIED RESEARCH

Augmented Neural Lyapunov Control

DAVIDE GRANDE[1,2], ANDREA PERUFFO[3], ENRICO ANDERLINI[1], AND GEORGIOS SALAVASIDIS[2]

[1]Department of Mechanical Engineering, University College London, WC1E 7JE London, U.K.
[2]National Oceanography Centre, SO14 3ZH Southampton, U.K.
[3]Mechanical, Maritime and Materials Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands

Corresponding author: Davide Grande (grande.rdev@gmail.com)

This work was supported in part by the National Oceanography Centre, Southampton, U.K.; in part by University College London, London, U.K.; and in part by the European Research Council through the SENTIENT Project under Grant ERC-2017-STG 755953.

ABSTRACT Machine learning-based methodologies have recently been adapted to solve control problems. The Neural Lyapunov Control (NLC) method is one such example. This approach combines Artificial Neural Networks (ANNs) with Satisfiability Modulo Theories (SMT) solvers to synthesise stabilising control laws and to prove their formal correctness. The ANNs are trained over a dataset of state-space samples to generate candidate control and Lyapunov functions, while the SMT solvers are tasked with certifying the correctness of the Lyapunov function over a continuous domain or by returning a counterexample. Despite the approach's attractiveness, issues can occur due to subsequent calls of the SMT module at times returning similar counterexamples, which can turn out to be uninformative and may lead to dataset overfitting. Additionally, the control network weights are usually initialised with pre-computed gains from state-feedback controllers, e.g. Linear-Quadratic Regulators. To properly perform the initialisation requires user time and control expertise. In this work, we present an *Augmented* NLC method that mitigates these drawbacks, removes the need for the control initialisation and further improves counterexample generation. As a result, the proposed method allows the synthesis of nonlinear (as well as linear) control laws with the sole requirement being the knowledge of the system dynamics. The ANLC is tested over challenging benchmarks such as the Lorenz attractor and outperformed existing methods in terms of successful synthesis rate. The developed framework is released open-source at: https://github.com/grande-dev/Augmented-Neural-Lyapunov-Control.

INDEX TERMS Computer-aided control design, Lyapunov methods, neural networks.

I. INTRODUCTION

One of the most common techniques used to analyse the stability of a dynamical system is Lyapunov's theory [1], [2]. Formally, an n-dimensional time-invariant dynamical system can be described as $\dot{x} = f(x, u)$, with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ representing the states and the control input respectively, with $f(x, u)$ denoting a continuous vector field defined over $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. The stability of an equilibrium point $x^\star$ can be inferred by means of a pseudo-energy function, known as a Lyapunov function (LF), that decreases as the state-space trajectories approach $x^\star$. When a LF is associated with a dynamical system with exogenous inputs, it is referred to as a Control Lyapunov function (CLF) [3]. Given a domain $\mathscr{D} \subseteq \mathbb{R}^n$, by defining a (C)LF as a function $V(x) : \mathscr{D} \to \mathbb{R}$, the stability of $x^\star$ can be assessed satisfying:

$$V(x^\star) = 0, \quad V(x) > 0, \quad \dot{V}(x, u) \leq 0, \tag{1}$$

where $\dot{V}(x, u) = \nabla V(x) \cdot f(x, u)$ represents the Lie derivative of V, and where the two inequalities must hold $\forall x \in \mathscr{D} \setminus \{x^\star\}$. Additionally, if $\dot{V}(x, u) < 0 \; \forall x \in \mathscr{D} \setminus \{x^\star\}$, the equilibrium is asymptotically stable. Without loss of generality, we consider $x^\star = 0$.

(C)LFs are notoriously difficult to design and no general method is known to synthesise them for nonlinear systems [1]. Several numerical methods have been proposed in the literature: notably, piecewise (linear or quadratic) and Sum-of-Squares LFs [4], [5], along with Genetic Algorithms [6], [7], or Artificial Neural Networks (ANNs) [2], [8], [9]. In the latter, the concept of *Lyapunov Neural Networks*

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiguang Feng.

takes shape, representing the use of a Feedforward ANN as a function approximator for LFs. However, sole use of numerical methods cannot guarantee the correctness of the (C)LFs that are generated. As a result, recent focus has been placed on providing a formal certification of (C)LFs' correctness [10], [11], [12]. These methods are based on a loop between a *Learner* and a *Falsifier*. The former, using a finite set of samples, trains a neural network that represents a candidate CLF; the latter checks whether the candidate CLF satisfies the conditions in (1) over $\mathscr{D}$. Advancements in formal techniques allow the verification of candidate CLFs as *certificate functions* [13], thanks to, e.g., *Satisfiability Modulo Theories* (SMT) solvers, capable of evaluating the correctness of a symbolic expression and of assessing whether a formula holds with respect to a set of constraints [14]. This class of solvers guarantees the *soundness* of the solution: when a logic expression is evaluated as *True*, the result is valid over the entire search domain [15]. SMTs guarantee that even if the training relies on a finite number of data points, the existence of a correct CLF can be formally certified, providing results equivalent to those of analytical proofs [11]. In this work we employ *dReal* for its capability to handle polynomials, trigonometric and exponential expressions [16]. Furthermore, when the SMT solvers verify that an expression does not hold true, a point violating the expression is returned. These points, referred to as counterexamples (CEs), are added to the training set to further assist the scouting of CLFs. The approach is therefore based on a learning-verifying paradigm with successive addition of CEs, known as CounterExample-Guided Inductive Synthesis (CEGIS) [11].

In line with [10], [11], [17], the twofold goal of this work lies in both finding a control policy and a stability certificate for nonlinear time-invariant systems, for which CLFs are notoriously difficult to obtain by analytical means. Following the seminal *Neural Lyapunov Control* (NLC) [10], several works synthesise (C)LFs employing neural architectures; from shallow [11], [12] to deep networks [17], [18], [19]. Further, activation functions vary significantly; e.g. ReLU, polynomial, tanh [10], [12], and softplus [17], [18].

There is rising interest in this method, as it finds applications in a wide range of applications: from guiding vehicles to their target destinations, e.g. boat motion, robotic and satellite tracking, to studying the stability of hybrid and switched systems. However, common drawbacks persist. Usually, the method exploits a pre-initialisation of the control law, normally through pre-computed Linear Quadratic Regulator (LQR) solutions, and the control gain is updated during the training to expand the Region Of Attraction (ROA). This initialisation limits the generality of the approach as it requires expert knowledge. Further, the neural training at times reaches a deadlock, and a periodic re-initialisation of the networks is necessary. The CEGIS paradigm can also suffer from overfitting subsets of sampled states when subsequent CEs are returned in a narrow state-space subset. This can lead to remarkable fluctuations of the loss function and in noticeably significant violations of the Lyapunov conditions [19]. This paper aims to address these three issues, namely the necessity for control weight initialisation, the periodic training deadlock and the dataset overfitting. The work specifically focuses on extending the learning abilities of the NLC algorithm through reducing the number of human inputs and pre-computations required.

Contributions: In this work, we propose novel solutions for the drawbacks identified when employing NLC methodology. Five significant elements are addressed: CEs generation logic, dataset composition and overfitting, algorithmic inefficiencies and sensitivity to learning rate. An upgraded Falsifier module mitigating the issues linked to the overfitting of clustered CEs and rendering their generation more efficient is proposed. Next, a sliding window to selectively disregard previously targeted old dataset points is devised, reducing the required computational burden. Finally, algorithmic refinements are introduced.

These improvements to the learning scheme help to overcome critical limitations, such as the need to pre-initialise the control gains and the periodic re-start of the neural network training. The architecture offers the possibility to design nonlinear control laws, and in so doing presenting another element of novelty. A modular framework that allows the testing of different ANN architectures and activation functions is designed and released open-source at: https://github.com/grande-dev/Augmented-Neural-Lyapunov-Control.

The remainder of this manuscript is structured as follows: Section II presents the CEGIS loop and its two components, i.e. the Learner and Falsifier, and Section III introduces the improvements to the state-of-the-art methods. Numerical tests are provided in Section IV, where our method is compared against the NLC. Finally, conclusions are outlined in Section V.

II. SYNTHESIS OF CONTROL LYAPUNOV FUNCTIONS

The following framework is built upon related literature, e.g. [10], [11], while the improvements are detailed in Section III. This method employs sequential iterations of the Learner and the Falsifier in order to *i)* tune the control gains while computing a candidate CLF, and *ii)* formally verify the stability of the resulting closed-loop system. While the Learner trains the ANNs, the Falsifier oversees the formal verification of the candidate CLF and returns possible counterexamples to the Learner. The overall procedure is illustrated in Fig. 1, where θ embeds the network parameters, $V_C(x)$ and $u_C(x)$ represent the candidate CLF and the candidate control law, respectively, and X the training dataset, initially populated with points generated from a Uniform distribution. Additionally, CE_{SMT} and CE_{DF} denote the CEs returned by the two verification modules that comprise the (Augmented) Falsifier.

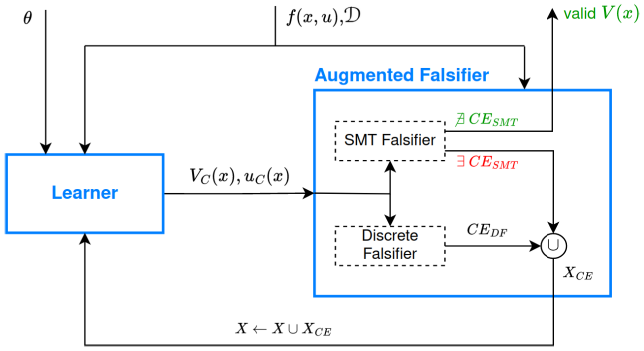


FIGURE 1. CEGIS learning method with Augmented Falsifier.

A. LEARNER

The Learner is the module tasked with training the ANNs to tune the control gains and to propose candidate CLFs. In this section, we discuss the ANNs architecture, the loss function and the tuning of the learning parameters.

1) CONTROL ARCHITECTURE

The control architecture proposed in this work is composed of three networks embodying the CLF, the linear and the nonlinear control laws. A user-defined, Boolean parameter, denoted *control-branch training selector* (ϕ), is employed to train either one of the two control ANNs. The resulting architecture is illustrated in Fig. 2. Naturally, a wider and deeper network ensures more flexibility and approximation power. On the other hand, this increases the computational training time and convolutes the symbolic expression of the CLF, typically representing a challenge for the Falsifier. Hence, the network architecture must be carefully selected, balancing generalisation power against simple expressions. To this end, we employ networks composed of two hidden layers that deliver CLFs consistently and rapidly, as shown in Section IV, following previous findings [11], [17], [18], [19].

2) LOSS FUNCTIONS

The loss function embodies a twofold objective; it drives the candidate CLF to satisfy the Lyapunov constraints, whilst maximising the ROA. As such, we devise the *Empirical* Lyapunov Risk Loss as:

$$L_{ELR} = \alpha_1 \sum_{i=1}^{N} \mathscr{R}(-V(x_i)) + \alpha_2 \sum_{i=1}^{N} \mathscr{R}(\dot{V}(x_i, u_i)) + \alpha_3 V(0)^2 + \alpha_4 \frac{1}{N} \sum_{i=1}^{N} (\|x_i\|_2 - \alpha_{ROA}\, V(x_i))^2 \quad (2)$$

where $\mathscr{R}(a) = ReLU(a) = \max(0, a)$ for a general input a, and where $\alpha_1, \ldots, \alpha_4, \alpha_{ROA}$ are tuning parameters and N is the cardinality of the dataset X (containing samples x_i). The first three terms account for the theoretical Lyapunov conditions in (1), while the final term's function is maximising the size of the ROA [10].

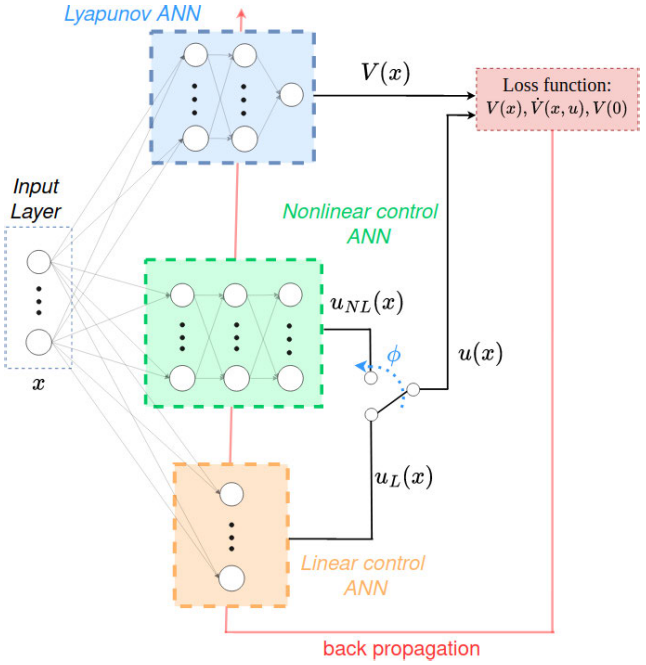


FIGURE 2. Augmented Neural Lyapunov Control architecture with Lyapunov ANN (blue box), nonlinear control ANN (green box) and linear control ANN (orange box). The red line represents the loss function back propagation, and ϕ the *control-branch training selector*.

By weighting the first three terms of the loss function higher than the fourth, the ROA tuning term can be considered as an additional side feature. Moreover, the fourth component induces a parabolic V shape, while α_{ROA} controls its steepness, with lower values of the latter tuning term corresponding to steeper CLFs, i.e. more aggressive tuning. When the fourth term in not employed, namely when maximising the ROA is not a priority, the values of α_1, α_2 and α_3 can be set up to 1.0, and $\alpha_4 = 0.0$, without any loss of generality (as shown in the case studies of Sec. IV). Additionally, when $V(0) = 0$ holds by construction, such as when the Lyapunov ANN has no bias and the activation function is zero in zero, α_3 can also be set to zero. To monitor whether the Lyapunov constraints alone are satisfied, we introduce the *Strict Lyapunov Risk Loss* (L_{SLR}):

$$L_{SLR} = \sum_{i=1}^{N} \mathscr{R}(-V(x_i)) + \sum_{i=1}^{N} \mathscr{R}(\dot{V}(x_i, u_i)) + V(0)^2 \quad (3)$$

which returns a positive value whenever any point in the dataset X violates (1). With the logic further detailed in Section III-A, this function is used to reduce the computational burden of the procedure by executing callbacks to the Falsifier only when the L_{SRL} is equal to zero.

The quantities $V(x_i)$ and $\dot{V}(x_i, u_i)$ in (2)-(3) can be readily evaluated. Let us define the output of the i-th layer as:

$$\boldsymbol{z_i} = \sigma_i(\boldsymbol{W_i}\, \boldsymbol{z_{i-1}} + \boldsymbol{b_i}),\ i = 1, \ldots, k \quad (4)$$

where $\boldsymbol{z_{i-1}}$ represents the input to the i-th layer, and $\boldsymbol{W_i}, \boldsymbol{b_i}, \sigma_i$ are the corresponding weight, bias and activation function,

respectively. A simple forward pass of the Lyapunov network returns $V(x_i)$, as:

$$V(x) = \sigma_k(\boldsymbol{W_k}\, \boldsymbol{z_{k-1}} + \boldsymbol{b_k}) \tag{5}$$

where k denotes the total number of layers of the network. The value of $\dot{V}(x_i, u_i)$ can be computed as:

$$\dot{V} = \left(\prod_{i=1}^{k} \text{diag}\left[\sigma'_{k-i+1}(\boldsymbol{z_{k-i}})\right] W_{k-i+1} \right) \cdot \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{6}$$

see [11] for a full derivation of (6). Conventionally, the element tasked with interfacing the numerical Learner and the Falsifier is referred to as *Translator* [12].

B. SMT FALSIFIER

The Falsifier module is designed to formally verify that a candidate function $V_c(x)$ is a CLF. This statement corresponds to evaluating the expression:

$$\forall x : (x \in \mathscr{D}) \Rightarrow (V_c(x^\star) = 0 \wedge V_c(x) > 0 \wedge \dot{V}_c(x, u) < 0) \tag{7}$$

This expression can be simplified: recall (5), and note that $V(0) = 0$ if we choose $\boldsymbol{b_i} = 0$ and activation functions such that $\sigma_i(0) = 0$ for all i. The falsification step is thus formulated as the logical negation of Eq. (7), meaning that the SMT solver searches for a solution of:

$$\exists x : (x \in \mathscr{D}) \Rightarrow (V_c(x) \leq 0 \vee \dot{V}_c(x, u) \geq 0) \tag{8}$$

If there exists such a point x, the candidate $V_c(x)$ does not satisfy the constraints (1), hence V_c is discarded and x is returned to the Learner as a counterexample. dReal is a δ-complete solver: this ensures that whenever V_c is deemed valid, then V_c is a CLF. However, it may return spurious counterexamples within a δ-error. While this may generate an infinite number of loops between Learner and Falsifier, it does *not* impair the correctness of the proposed procedure. The δ precision is problematic when checking the constraints (1) within a neighborhood of the origin. The exclusion of a small neighborhood of the origin from the verification step is proposed as mitigation [10], [11], introducing a lower boundary on the solutions domain. In the case of a spherical domain, the domain is defined as: $\underline{\gamma} \leq \|x\|_2 \leq \overline{\gamma}$, for given radii $\underline{\gamma}$ and $\overline{\gamma}$. As such, our method guarantees the *practical* stability of the underlying model [11], formally guaranteeing that the state remains bounded within $\underline{\gamma}$ upon convergence.

III. AUGMENTED NEURAL LYAPUNOV CONTROL

In this section, specific limitations of the original NLC method are mitigated. The proposed upgraded procedure is hereby defined as *Augmented* NLC (ANLC).

A. AUGMENTED FALSIFIER

At each callback of the SMT Falsifier, one single CE is identified, and a point cloud in the vicinity of the CE returned. The SMT callback is generally time-consuming, causing a bottleneck in the procedure. To overcome this issue, a numerical unit, called *Discrete Falsifier* (DF), numerically (i.e. rapidly) generates CEs before the SMT call.

The learning loop evolves as follows. The learning iterations, while minimising L_{ELR}, cease when $L_{SRL} = 0$ is attained. When this occurs, the candidate CLF satisfies the constraints (1) over all samples. Next the DF is called. The domain is discretised with a user-defined precision and the values of $V(x)$ and $\dot{V}(x, u)$ are evaluated over all the grid points. If points violating (1) are found, these are stored in the set CE_{DF}, added to the dataset and the training resumes. If the cardinality of CE_{DF} exceeds a threshold ζ_{DF}, a randomly selected subset of cardinality ζ_{DF} is added to the dataset. In the case of no CEs being obtained, the SMT is invoked to *formally* verify the correctness of the candidate. Finally, if the SMT locates a CE, a set containing ζ_{SMT} new points are added to the dataset (defined as X_{CE} in Fig. 1), otherwise $V_c(x)$ is verified to be a CLF.

B. NETWORK-SPECIFIC LEARNING RATE AND SCHEDULER

The Learning Rate (LR) has a significant impact on the learning ability of ANNs [20], [21]. Low values of LR often lead to the loss function getting stuck in local minima, while high values typically result in unstable training [22]. In [10], both the Lyapunov and the control ANNs use a single LR, and at times this leads to training stalls. In this work, the Lyapunov and control LRs are separated, and referred to as λ and λ_c, respectively. Additionally, to resemble the re-initialisation of the ANN weights, a cosine annealing scheduler is added [23]. This scheduler cyclically oscillates the LR between a minimum and maximum value, allowing it to fluctuate over a range while briefly employing both conservative and aggressive values. This element avoids hard reset of the weight at pre-defined intervals, mitigating the risk of losing the learning progress in the midst of the training process.

C. ALGORITHM

The scheme of the ANLC method is reported in Algorithm 1, where the *timeout* defines a time threshold for the SMT Falsifier to compute the CEs.

D. COUNTEREXAMPLE SELECTION

The CEGIS paradigm gives rise to a dynamic dataset that gradually increases in size as successive CEs are returned by the Falsifier. As the CEGIS loops progress, the algorithm slows down, since the computational burden derived from the training of the ANNs depends, among other parameters, on the size of the dataset. Note that, experimentally, the SMT often returns similar CEs at successive iterations, i.e. a significant portion of the dataset clusters in a small region of the state-space. This event is denoted *counterexample overfitting*, as illustrated in Fig. 3.

In this work, we introduce a *selective sliding window* as a means of mitigation. The initial dataset X_I, which contains samples scattered over the entire domain, is held unchanged.

Algorithm 1 Augmented Neural Lyapunov Control

function Learner(X, f, ANN_θ)
  repeat
    $V_\theta(x), u_\theta(x) \leftarrow ANN_\theta(x)$ ▷ ANN forward pass
    $\dot{V} \leftarrow$ Translator
    Compute loss L_{ELR}, L_{SLR}
    $\theta \leftarrow \theta - \nabla_\theta L_{ELR}$ ▷ Update weights
  until ($L_{SLR} > 0$)
  return $V_\theta(x), u_\theta(x)$
function Discrete Falsifier($V_\theta^S, \dot{V}_\theta^S, \zeta_{DF}$)
  Discretise $\mathscr{D}$
  Numerically evaluate ($V_\theta^S \leq 0, \dot{V}_\theta^S \geq 0$)
  $CE_{DF} \leftarrow$ Violations points (max size ζ_{DF})
  return CE_{DF}
function Main()
  Input: f, X, ζ_{DF}, ζ_{SMT}, $\mathscr{D}$, $\underline{\gamma}$, $\overline{\gamma}$, θ, λ, λ_c, ϕ, $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_{ROA}$, optional control gains (q^{LQR})
  repeat
    if (size(X) $\geq X_{max}$): Apply sliding window
    $V_\theta(x), u_\theta(x) \leftarrow$ Learner(X, f, ANN_θ)
    Compute symbolic values $f_\theta^S, u_\theta^S, V_\theta^S, \dot{V}_\theta^S$
    if ($L_{SLR} == 0$) **then**
      $CE_{DF} \leftarrow$ Discr. Falsifier($V_\theta^S, \dot{V}_\theta^S, \zeta_{DF}$)
      if CE_{DF} is `None` **then**
        $CE_{SMT} \leftarrow$ SMT Falsifier($\cdot$)
      $X_{CE} \leftarrow (CE_{DF} \cup CE_{SMT})$
      if (not `sat`**):** $X \leftarrow (X \cup X_{CE})$
  until not (*converged* **or** *timeout*)

A sliding window is applied to the counterexample dataset X_{CE}. When a maximum size is attained, the least recent CEs are deleted in order to keep the dataset cardinality bounded, as illustrated in Fig. 4. This element limits the risk of CE overfitting – already known as a possible cause of reduced algorithm efficiency [19] while the fixed dataset dimension prevents the training performance from deteriorating over successive loops.

IV. EXPERIMENTAL EVALUATION

We first compare our procedure against the work of [10] over an inverted pendulum system and later challenge it over a Lorenz attractor benchmark. The results reported in the following section highlight the improvements gained when compared to the original approach. The software is implemented in *Python* v3.7, with *dReal* v4.21.6 and *Pytorch* v1.4.0. The runs are executed on a standard office laptop computer (8 CPUs at 1.90GHz, no GPU).

A. CONTROL SYSTEM WITHOUT INITIALISATION

The problem of the stabilisation of an inverted pendulum, broadly discussed in [10] and [17], is used as a benchmark test. In these works, the control ANN is initialised with a pre-computed LQR law. Hereby, the effect of not initialising the ANN weights is quantified. The inverted pendulum

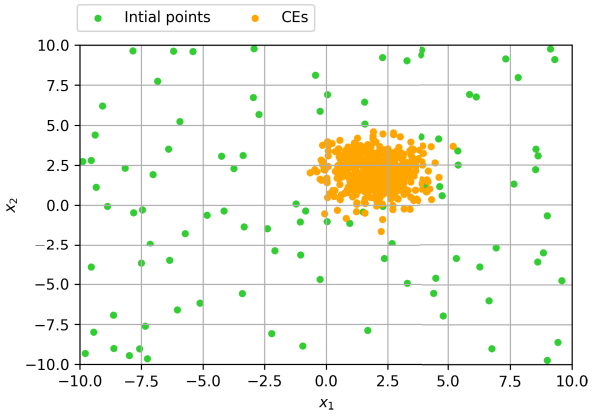


FIGURE 3. Dataset with *initial points* (X_I) and clustered counterexamples.

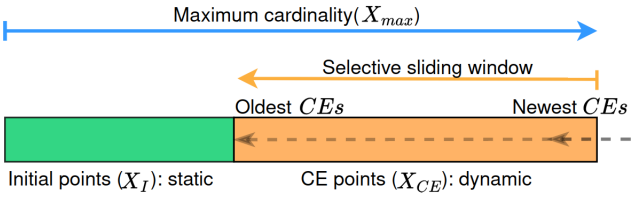


FIGURE 4. Dataset with *selective sliding window* logic, with $X = X_I \cup X_{CE}$.

TABLE 1. ANLC: campaign parameters.

$\underline{\gamma}$ [deg]	$\overline{\gamma}$ [deg]	δ	N	λ	λ_c
5.73	343	10^{-6}	500	[0.0, 0.01]	[0.0, 1.0]

dynamics can be written as follows:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = J_z^{-1}(mgl \sin x_1 - bx_2 + u) \end{cases} \tag{9}$$

where x_1, x_2 represent the angular position and velocity of the mass respectively, and u the control input. The parameters b, l, m, J_z denote the drag coefficient, the length of the pendulum arm, the lumped mass and the moment of inertia respectively. Table 1 collects the test parameters, as per [10], [17], where λ, λ_c are reported in terms of the minimum and maximum values of the cosine annealing scheduler. In the ANLC tests, the loss function gains are, intentionally, not finely tuned (i.e. $\alpha_{1,2,3} = 1$ and $\alpha_4 = 0$) and each hidden layer is composed of 10 neurons. The activation functions are linear, quadratic and linear for the two hidden and output layers, respectively.

To compare the performance of the NLC and the ANLC algorithms, a simulation campaign composed of six case scenarios is designed and the results are reported in Table 2. For each scenario, 50 tests are run (with different seeds). The results are presented in terms of iterations required for the algorithms to converge and reported as per mean value (μ) and three standard deviation (3σ). A test is defined as *converged* when a stabilising control law and a CLF are obtained within a prescribed threshold of maximum number of learning iterations.

TABLE 2. Sensitivity to control weights initialisation.

Algo.	Control pre-initialised	Weights re-initialised	Iterations $[\mu(3\sigma)]$	Converged tests [%]
NLC	Yes	Yes	1107(2239)	78
NLC	No	Yes	2000(0)	0
NLC	Yes	No	698(954)	52
ANLC	Yes	No	478(443)	84
NLC	No	No	1000(0)	0
ANLC	No	No	464(1120)	60

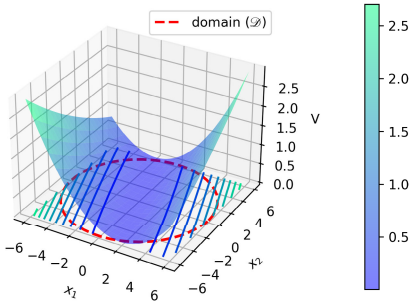


FIGURE 5. CLF inverted pendulum.

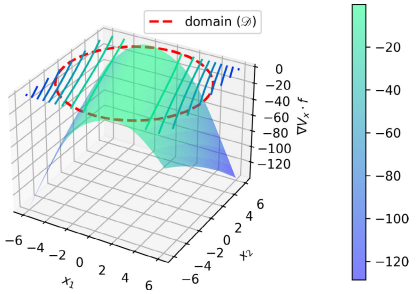


FIGURE 6. Lie derivative inverted pendulum.

Two initial NLC scenarios are run for a maximum of 2×1000 iteration. If the scenario has not converged to a solution within the first 1000 iterations, the weights of the ANNs are then re-initialised and the scenario is run once more for up to 1000 more iterations. When the control network is not initialised with an LQR, the performance of the algorithm drops from 78% of successful tests to zero. This clearly highlights the control initialisation as a key part of the algorithm.

The next four cases compare the performance of the ANLC algorithm against that of the NLC algorithm with focus on the control weight initialisation. Again, the performance of the NLC algorithm is reduced when the control network is not initialised (from 52% to 0%). The ANLC method outperforms the NLC when the control is initialised (84% vs. 52%), and even the NLC with re-initialisation of the weights, (84% vs. 78%), despite being run for half of the iterations and not initialised (60% vs. 0%). The CLF synthesised in one of the ANLC tests, obtained by selecting $V = \boldsymbol{W_3}(\boldsymbol{W_2}(\boldsymbol{W_1}\boldsymbol{x}))^2$ as the architecture, is illustrated in Fig. 5, with the corresponding Lie derivative function in Fig. 6.

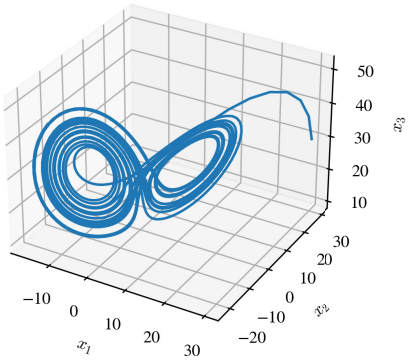


FIGURE 7. Open-loop Lorenz system trajectory.

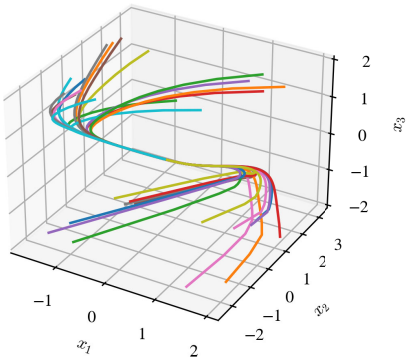


FIGURE 8. Closed-loop Lorenz system trajectories.

B. CONTROLLED LORENZ SYSTEM

To investigate how the framework performs with a more complex 3-dimensional system and when employing *nonlinear control laws*, the Lorenz equations are selected. The former represent a simplified model of highly nonlinear and coupled phenomena of atmospheric convection [24]. Let us consider the *controlled Lorenz system* [25], [26], i.e. the system described by:

$$\begin{cases} \dot{x}_1 = -\sigma(x_1 - x_2) + u_1 \\ \dot{x}_2 = r x_1 - x_2 - x_1 x_3 + u_2 \\ \dot{x}_3 = x_1 x_2 - b x_3 + u_3 \end{cases} \tag{10}$$

where $\mathbf{x} = [x_1, x_2, x_3]^T$ represents the state-space vector and $\mathbf{p} = [\sigma, r, b]^T$ the scalar parameters. By selecting $\sigma = 10$, $b = 8/3$ and $r = 28$, as in [25] and [26], the origin can be rendered as an unstable equilibrium and the characteristic butterfly-like *strange attractor* (or *Lorenz attractor*) obtained, as reported in the open-loop trajectory of Fig. 7. For this study, a nonlinear control law is selected by employing *softplus* activation functions, and two hidden layers of 8 neurons each, while the Lyapunov ANN is chosen as $V = \boldsymbol{W_3}(\boldsymbol{W_2}(\boldsymbol{W_1}\boldsymbol{x})^2)$.

Out of ten tests run with different seeds, seven converged within 1000 iterations (maximum computational time of 352 $[s]$). The evolution of $\dot{V}(\mathbf{x}, \mathbf{u})$, plotted in the (x_1, x_2)-plane (with x_3 set to zero), is reported in Fig. 9 and in Fig. 10, corresponding to the first and last training iterations

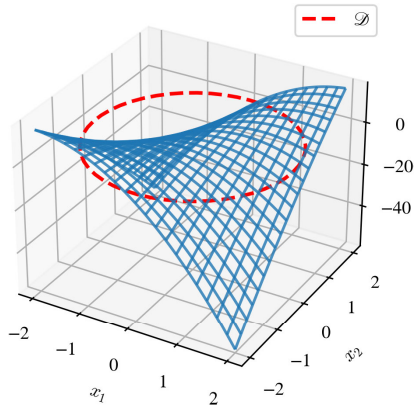


FIGURE 9. Lie derivative Lorenz (x_1, x_2)-plane, at the first training iteration.

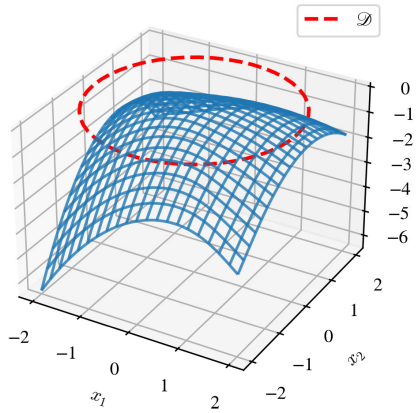


FIGURE 10. Lie derivative Lorenz (x_1, x_2)-plane, at the last training iteration.

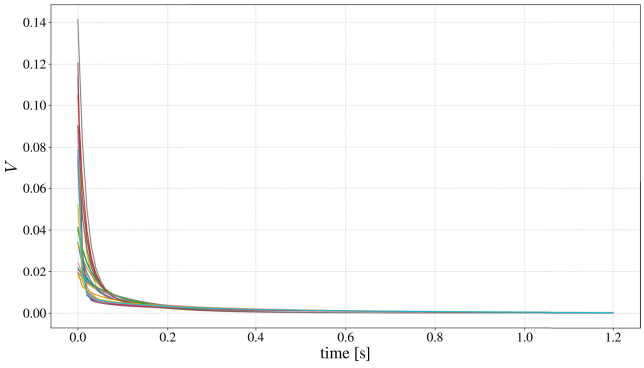


FIGURE 11. Lyapunov values along Lorenz system trajectories.

respectively. Note that the Lie derivative is initially non-negative due to the random ANNs initialisation, and evolves into a negative-definite function upon convergence. Figure 8 shows 30 closed-loop trajectories starting from randomly selected initial points, while Fig. 11 highlights the corresponding fast decreasing Lyapunov values as the solutions approach the equilibrium state. The control function values over time corresponding to one converged controller are reported in Fig. 12.

Experimentally, we can note that the synthesis of CLF with nonlinear control laws is computationally more demanding and takes longer to terminate, both on the Learner side,

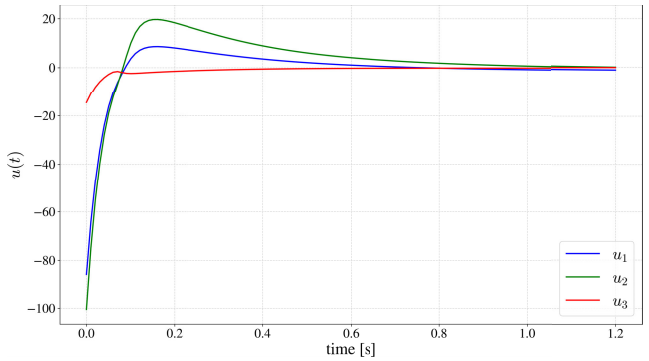


FIGURE 12. Control function values over time.

as more iterations are needed for the convergence of the training, and on the Falsifier side, as a more complex symbolic expression must be evaluated. Hence, we highlight that a trade off must be made between the generalisation power of the network and computational time, in particular when higher dimensional systems are analysed. This would equate to more than 10 dimensions, as per previous findings [18].

V. CONCLUSION

In this work a procedure, which is robust to well-known issues, to inductively synthesise CLFs, by devising an upgraded Falsifier and careful selection of useful counterexamples, has been presented. The Augmented NLC method is shown to be capable of computing stabilising control laws without the need to pre-initialise the control gains, further increasing the generalisation power and overall applicability of this method. Both linear and nonlinear control laws are synthesised for unstable dynamics, showing the modularity of the proposed architecture, while limitations are highlighted. This method finds applications in a variety of nonlinear control problems, from guiding unmanned vehicles to robotics problems, e.g. landing rockets. Future work should focus on scalability to both higher dimensional and uncertain dynamics, issues that will require dedicated assessment. Application to the control of robotic systems, such as Autonomous Underwater Vehicles, is a matter of ongoing research effort.

ACKNOWLEDGMENT

The authors would like to thank K. Soonho, S. Gao, G. Thomas, D. Riccio, G. Corsini, Y. Liu and A. B. Philips for their valuable insights and support. In addition, the authors thank K. Yearwood for proofreading this manuscript.

REFERENCES

[1] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*, vol. 199, no. 1. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[2] N. Noroozi, P. Karimaghaee, F. Safaei, and H. Javadi, "Generation of Lyapunov functions by neural networks," in *Proc. World Congr. Eng.*, 2008, pp. 1–5.

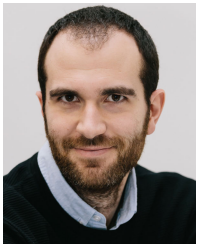
[3] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, vol. 6. Springer, 2013.

[4] J. Anderson and A. Papachristodoulou, "Advances in computational Lyapunov analysis using sum-of-squares programming," *Discrete Continuous Dyn. Syst. B*, vol. 20, no. 8, pp. 2361–2381, Aug. 2015.

[5] A. Papachristodoulou and S. Prajna, "On the construction of Lyapunov functions using the sum of squares decomposition," Presented at the 41st IEEE CDC, Dec. 2022.
[6] M. Sabouri, P. Setoodeh, and M. H. Asemani, "Construction of Lyapunov functions using multi-objective genetic algorithm," Presented at the 28th IEEE ICEE, Aug. 2020.
[7] C. F. Verdier and M. Mazo Jr., "Formal controller synthesis via genetic programming," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7205–7210, Jul. 2017, doi: 10.1016/j.ifacol.2017.08.1362.
[8] D. V. Prokhorov, "A Lyapunov machine for stability analysis of nonlinear systems," Presented at the 28th IEEE ICEE, Jun./Jul. 1994.
[9] G. Serpen, "Empirical approximation for Lyapunov functions with artificial neural nets," Presented at the IEEE IJCNN, Jul./Aug. 2005.
[10] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," Presented at the NeurIPS, 2019.
[11] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Formal synthesis of Lyapunov neural networks," *IEEE Control Syst. Lett.*, vol. 5, no. 3, pp. 773–778, Jul. 2021, doi: 10.1109/LCSYS.2020.3005328.
[12] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, and A. Peruffo, "FOSSIL: A software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks," Presented at the HSCC, May 2021.
[13] 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 Trans. Robot.*, vol. 39, no. 3, pp. 1749–1767, Jun. 2023, doi: 10.1109/TRO.2022.3232542.
[14] R. Sebastiani, "Lazy satisfiability modulo theories," *J. Satisfiability, Boolean Model. Comput.*, vol. 3, nos. 3–4, pp. 141–224, Dec. 2007, doi: 10.3233/SAT190034.
[15] S. Gao, J. Kapinski, J. Deshmukh, N. Roohi, A. Solar-Lezama, N. Arechiga, and S. Kong, "Numerically-robust inductive proof rules for continuous dynamical systems," Presented at the CAV, 2019.
[16] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," Presented at the CADE, 2013.
[17] R. C. B. Rego and F. M. U. Araújo, "Learning-based robust neuro-control: A method to compute control Lyapunov functions," *Int. J. Robust Nonlinear Control*, vol. 32, no. 5, pp. 2644–2661, Mar. 2022, doi: 10.1002/rnc.5399.
[18] L. Grüne, "Computing Lyapunov functions using deep neural networks," 2020, *arXiv:2005.08965*.
[19] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," Presented at the RSS, 2021.
[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[21] G. Montavon, G. Orr, and K.-R. Müller, *Neural Networks: Tricks of the Trade*, vol. 7700. Springer, 2012.
[22] Đ. Žikelić, M. Lechner, K. Chatterjee, and T. A. Henzinger, "Learning stabilizing policies in stochastic control systems," Presented at the SRML, 2022.
[23] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, *arXiv:1608.03983*.
[24] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmos. Sci.*, vol. 20, no. 2, pp. 130–141, 1963, doi: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.
[25] R. Choroszucha, "Control of the Lorenz equations," Dept. Nav. Mar. Eng., Univ. Michigan, Ann Arbor, MI, USA, Tech. Rep., 2000.
[26] P. P. C. Alzate, G. C. Velez, and F. Mesa, "Chaos control for the Lorenz system," *Adv. Stud. Theor. Phys.*, vol. 12, no. 4, pp. 181–188, 2018, doi: 10.12988/astp.2018.8413.

DAVIDE GRANDE received the B.Sc. and M.Sc. degrees in control engineering from Politecnico di Milano, Italy, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree in mechanical engineering with University College London, U.K.

From 2017 to 2018, he worked with INESCTEC, Portugal, on the modeling and control of underwater vehicles. From 2018 to 2020, he was with Airbus Defence and Space, U.K., supporting the design of guidance and attitude control systems for European and international space missions. His current research interests include control systems applied to autonomous vehicles and machine learning-based fault tolerant control.

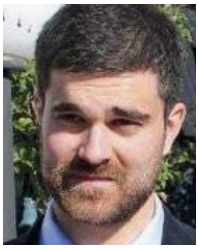


ANDREA PERUFFO received the D.Phil. degree in computer science from the University of Oxford, and the Laurea degree in information engineering and the M.S. degree in automation engineering from the University of Padua (IT), in 2012 and 2015, respectively. He is a postdoc at the Delft Center for Systems and Control.

He was a Research Engineer with Inria Rennes, in 2016, and visited the Erato MMSD Group, NII, Tokyo, in 2020. His current research interests include hybrid systems, formal verification, and control theory.

ENRICO ANDERLINI received the D.Eng. degree in offshore renewable energy from the Industrial Doctoral Centre for Offshore Renewable Energy, a partnership of the University of Edinburgh, University of Exeter, and University of Strathclyde, U.K., in 2018.

He was a Senior Research Fellow with the Department of Mechanical Engineering, University College London, U.K. He is currently a Product Manager with Manz, Sasso Marconi, Italy, where he is focusing on the field of manufacturing of lithium-ion batteries, and a Visiting Researcher with University College London. His current research interests include dynamics and control theory, focusing on learning-based methods.



GEORGIOS SALAVASIDIS received the integrated B.Sc. and M.Sc. degrees (Hons.) in electrical and computer engineering from the Democritus University of Thrace, Greece, in 2013, and the Ph.D. degree in marine robotics from the University of Southampton and the National Oceanography Centre, U.K., in 2019, a research funded by the European Commission through the Marie Sklodowska-Curie Initial Training Network (ROBOCADEMY).

He is currently a Robotics Researcher with the Marine Autonomous and Robotic Systems (MARS) Group, U.K. National Oceanography Centre. His current research interests include the entire perception–action–learning loop of marine autonomous systems, including control, perception, navigation, decision-making, and networked robot operations, with a particular focus on enabling persistent presence and long-range operations of autonomous underwater vehicles in remote GPS-denied environments, such as the Arctic Ocean and the Antarctic.