EPiC Series in Computing
Volume 90, 2022, Pages 142–184
Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)

ARCH-COMP22 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants

Diego Manzanas Lopez[1], Matthias Althoff[5], Luis Benet[2], Xin Chen[6], Jiameng Fan[8], Marcelo Forets[3], Chao Huang[7], Taylor T. Johnson[1], Tobias Ladner[5], Wenchao Li[8], Christian Schilling[4], Qi Zhu[9],

[1] Vanderbilt University
Nashville, TN
{diego.manzanas.lopez, taylor.johnson}@vanderbilt.edu
[2] Instituto de Ciencias Físicas, Universidad Nacional Autónoma de México (UNAM), México
benet@icf.unam.mx
[3] Universidad de la República, Montevideo, Uruguay
mforets@gmail.com
[4] Aalborg University, Aalborg, Denmark
christianms@cs.aau.dk
[5] Technische Universität München (TUM), Munich, Germany
{althoff, tobias.ladner}@tum.de
[6] University of Dayton, Dayton OH, USA
xchen4@udayton.edu
[7] University of Liverpool, Liverpool, UK
chao.huang2@liverpool.ac.uk
[8] Boston University, Boston, USA
{jmfan, wenchao}@bu.edu
[9] Northwestern University, Evanston, USA
qzhu@northwestern.edu

Abstract

This report presents the results of a friendly competition for formal verification of continuous and hybrid systems with artificial intelligence (AI) components. Specifically, machine learning (ML) components in cyber-physical systems (CPS), such as feedforward neural networks used as feedback controllers in closed-loop systems are considered, which is a class of systems classically known as intelligent control systems, or in more modern and specific terms, neural network control systems (NNCS). We more broadly refer to this category as AI and NNCS (AINNCS). The friendly competition took place as part of the workshop Applied Verification for Continuous and Hybrid Systems (ARCH) in 2022. In the fourth edition of this AINNCS category at ARCH-COMP, four tools have been applied to solve 10 different benchmark problems. There are two new participants: CORA and POLAR, and two previous participants: JuliaReach and NNV. The goal of this report is to be a snapshot of the current landscape of tools and the types of benchmarks for which these tools are suited. The results of this iteration significantly outperform those of any previous year, demonstrating the continuous advancement of this community in the past decade.

1 Introduction

Neural Networks (NNs) have demonstrated an impressive ability for solving complex problems in numerous application domains [57]. The success of these models in contexts such as adaptive control, non-linear system identification [41], image and pattern recognition, function approximation, and machine translation, has stimulated the creation of technologies that are directly impacting our everyday lives [50], and has led researchers to believe that these models possess the power to revolutionize a diverse set of arenas [46].

Despite these achievements, there have been reservations in utilizing them within high-assurance systems for a variety of reasons, such as their susceptibility to unexpected and errant behavior caused by slight perturbations in their inputs [36]. In a study by Szegedy et al. [51], the authors demonstrated that by carefully applying a hardly perceptible modification to an input image, one could cause a successfully trained neural network to produce an incorrect classification. These inputs are known as *adversarial examples*, and their discovery has caused concern over the safety, reliability, and security of neural network applications [57]. As a result, there has been a large research effort directed towards obtaining an explicit understanding of neural network behavior.

Neural networks are often viewed as “black boxes,” whose underlying operation is often incomprehensible, but the last several years have witnessed numerous promising white-box verification methods proposed towards reasoning about the correctness of their behavior. However, it has been demonstrated that neural network verification is an NP-complete problem [33], and while current state-of-the-art verification methods have been able to deal with small networks, they are incapable of dealing with the complexity and scale of networks used in practice ([37, 21, 6]). Additionally, while in recent years there has been a large amount of work focused on verifying pre-/post-conditions for neural networks in isolation, reasoning about the behavior of their usage in cyber-physical systems, such as in neural network control systems, remains a key challenge.

The following report aims to provide a survey of the landscape of the current capabilities of verification tools for *closed-loop systems with neural network controllers*, as these systems have displayed great utility as a means for learning control laws through techniques such as reinforcement learning and data-driven predictive control [19, 53]. Furthermore, this report aims to provide readers with a perspective of the intellectual progression of this rapidly growing field and stimulate the development of efficient and effective methods capable of use in real-life applications.

Disclaimer The presented report of the ARCH-COMP friendly competition for *closed-loop systems with neural network controllers*, termed in short AINNCS (Artificial Intelligence / Neural Network Control Systems), aims to provide the landscape of the current capabilities of verification tools for analyzing these systems that are classically known as *intelligent control systems*. This AINNCS ARCH-COMP category is complementary to the ongoing Verification of Neural Networks Competition (VNN-COMP) [9], the latter of which focuses on open-loop specifications of neural networks, while the AINNCS category focuses on closed-loop behaviors of dynamical systems incorporating neural networks. We would like to stress that each tool has unique strengths—not all of the specificities can be highlighted within a single report. To reach a consensus in what benchmarks are used, some compromises had to be made so that some tools may benefit more from the presented choice than others. To establish further trustworthiness of the results, the code with which the results have been obtained is publicly available at gitlab.com/goranf/ARCH-COMP.

Specifically, this report summarizes results obtained in the 2022 friendly competition of the ARCH workshop[1] for verifying systems of the form

$$\dot{x}(t) = f(x(t), u(x,t)),$$

where $x(t)$ and $u(x,t)$ correspond to the states and inputs of the plant at time t, respectively, and where $u(x,t)$ is the output of a feedforward neural network provided an input of the plant state x at time t. This year is the fourth iteration of the AINNCS category at ARCH-COMP and builds on the previous iterations and reports from 2019, 2020, and 2021 [38, 31, 30]. Participating tools are summarized in Sec. 2. Please, see [57] for further details on these and additional tools. The results of our selected benchmark problems are shown in Sec. 3 and are obtained on the tool developers' own machines. Thus, one has to factor in the computational power of the processors used, summarized in Appendix A, as well as the efficiency of the programming language of the tools. The architecture of the closed-loop systems we will evaluate is depicted in Figure 1, where the input to the NN controller is additionally sampled.

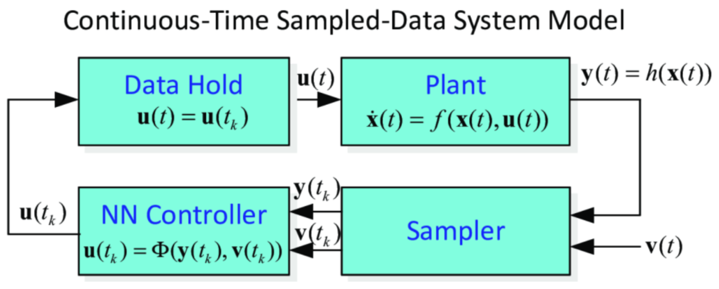

Continuous-Time Sampled-Data System Model
Data Hold
$\mathbf{u}(t)=\mathbf{u}(t_k)$
$\mathbf{u}(t)$
Plant
$\dot{\mathbf{x}}(t)=f(\mathbf{x}(t),\mathbf{u}(t))$
$\mathbf{y}(t)=h(\mathbf{x}(t))$
$\mathbf{u}(t_k)$
NN Controller
$\mathbf{u}(t_k)=\Phi(\mathbf{y}(t_k),\mathbf{v}(t_k))$
$\mathbf{y}(t_k)$
$\mathbf{v}(t_k)$
Sampler
$\mathbf{v}(t)$

Figure 1: Closed-loop architecture of the benchmarks to be verified.

The goal of the friendly competition is not to rank the results, but rather to present the landscape of existing solutions in a breadth that is not possible with scientific publications in classical venues. Such publications would typically require the presentation of novel techniques, while this report showcases the current state-of-the-art tools. The selection of the benchmarks has been conducted within the forum of the ARCH website (cps-vo.org/group/ARCH), which is visible for registered users and registration is open for anyone.

[1]Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), cps-vo.org/group/ARCH

2 Participating Tools

We present a brief overview of all the participating tools in this friendly competition. The tools are CORA, JuliaReach, NNV, and POLAR. The tools participating in the *Artificial Intelligence / Neural Network Control Systems in Continuous and Hybrid Systems Plants* (AINNCS) category are introduced subsequently in alphabetical order.

JuliaReach (Luis Benet, Marcelo Forets, Christian Schilling) JuliaReach [15] is an open-source software suite for reachability computations of dynamical systems, written in the Julia language and available at `http://github.com/JuliaReach`. The package ClosedLoopReachability.jl handles the closed-loop analysis and queries sub-problems to our other library ReachabilityAnalysis.jl for continuous-time analysis of plant models. Additional set computations are performed with LazySets.jl [24]. The algorithm we use is described in [47]. For the plant analysis we use the sound algorithm `TMJets` based on interval arithmetic and Taylor models, which is implemented in Taylor models [11, 14], which itself integrates TaylorSeries.jl [12, 13] and TaylorIntegration.jl [42]. The algorithm is a form of jet transportation using a Taylor polynomial with interval coefficients and uses the following main parameters for tweaking: the absolute tolerance `abstol` and two parameters to define the order at which the Taylor expansion is cut in time (`orderT`) resp. in space (`orderQ`). For neural-network analysis we use an abstract interpretation based on zonotopes [49]. For falsification, JuliaReach chooses an initial point but uses set-based analysis since, although most models are deterministic, non-validated simulations may yield wrong results.

NNV NNV (Neural Network Verification Tool) [55, 52, 54, 60, 62, 59, 58, 56, 61, 1] is a Matlab toolbox that implements reachability analysis methods for neural network verification, with a particular focus on applications of closed-loop neural network control systems in autonomous cyber-physical systems. NNV uses a star-set state-space representation and reachability algorithm that allows for a layer-by-layer computation of exact or overapproximate reachable sets for feed-forward and convolutional neural networks. The star-set based algorithm is naturally parallelizeable, which allowed NNV to be designed to perform efficiently on multi-core platforms. Additionally, in the event that a particular safety property is violated, NNV can be used to construct and visualize the complete set of counterexample inputs for a neural network. Using NNV in combination with HyST [8, 7] and CORA [2, 3, 4] allows for the verification of closed-loop neural network control systems with nonlinear plant dynamics. The tool along with all of the relevant experiments and publications can be found at `https://github.com/verivital/nnv`.

CORA The COntinuous Reachability Analyzer (CORA) [2] is a collection of MATLAB classes for the formal verification of cyber-physical systems using reachability analysis and is available at `https://cora.in.tum.de`. CORA integrates various vector and matrix set representations and operations on them as well as reachability algorithms of various dynamic system classes. For this competition, we used the approach described in [35] for open-loop and closed-loop neural network verification based on polynomial zonotopes [34]. Polynomial zonotopes are particularly well suited for the verification of neural networks due to their polynomial time complexity on many operations. The approach described in [35] realizes a fast layer-based computation of an over-approximation of the output set of networks with various activation functions, including ReLU, sigmoid, and tanh. Our neural network verification approach is naturally integrated in our reachability analysis methods for linear and nonlinear plant dynamics.

POLAR POLAR (**POL**ynomial **AR**ithmetic-based framework) [26] is a C++ open-source tool for efficient time-bounded reachability analysis of neural-network controlled systems, which is available at https://github.com/ChaoHuang2018/POLAR_Tool. The tool is implemented based on the GNU open-source libraries and the C++ library of Flow* [16]. The purpose of POLAR is **not** to compute range-overeapproximations for NNCS reachable sets. It seeks to compute **functional overapproximations** for the flowmap that defines all executions of the system. The tool enables precise layer-by-layer propagation of Taylor Models (TMs) for general feed-forward neural networks. The basic TM arithmetic cannot handle ReLU that is non-differentiable (cannot produce the polynomial), and also suffers from low approximation precision (large remainder). POLAR addresses these challenges through a novel use of *univariate Bernstein polynomial approximation* and *symbolic remainders*. Motivated by ReachNN* [27, 23, 22], univariate Bernstein polynomial approximation enables the handling of non-differentiable activation functions and local refinement of Taylor models. Symbolic remainders which is originally presented in [17] can taper the growth of interval remainders by avoiding the so-called wrapping effect [29] in linear mappings.

3 Benchmarks

For the competition, we have selected 10 benchmarks, 3 more than last iteration, increasing the complexity of the competition. A few of them, such as the TORA benchmark, are presented with several different controllers that can be analyzed. We now describe these benchmarks in no particular order and we have made them readily available online.[2] All benchmarks are derived for continuous time. Given the continuous dynamics $\dot{x} = f(x)$, where $x \in \mathbb{R}^n$ is the state vector, the discrete-time versions for a time increment of Δt are obtained in this competition using forward Euler integration:

$$x(k+1) = x(k) + f(x)\Delta t.$$

3.1 Adaptive Cruise Controller (ACC)

The Adaptive Cruise Control (ACC) benchmark is a system that tracks a set velocity and maintains a safe distance from a lead vehicle by adjusting the longitudinal acceleration of an ego vehicle [40]. The neural network computes optimal control actions while satisfying safe distance, velocity, and acceleration constraints using model predictive control (MPC) [44]. For this case study, the ego car is set to travel at a set speed $v_{set} = 30$ and maintains a safe distance D_{safe} from the lead car. The car's dynamics are described by the following equations [53, p. 17]:

$$\begin{aligned} \dot{x}_{\text{lead}}(t) = v_{\text{lead}}(t), \dot{v}_{\text{lead}}(t) = a_{\text{lead}}(t), \dot{a}_{\text{lead}}(t) = -2a_{\text{lead}}(t) + 2a_{c,\text{lead}} - uv_{\text{lead}}^2(t), \\ \dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t), \dot{v}_{\text{ego}}(t) = a_{\text{ego}}(t), \dot{a}_{\text{ego}}(t) = -2a_{\text{ego}}(t) + 2a_{c,\text{ego}} - uv_{\text{ego}}^2(t), \end{aligned} \tag{1}$$

where x_i is the position, v_i is the velocity, a_i is the acceleration of the car, $a_{c,i}$ is the acceleration control input applied to the car, and $u = 0.0001$ is a coefficient for air drag, where $i \in \{\text{ego}, \text{lead}\}$. For this benchmark we evaluate a neural network controller with five layers and 20 neurons each. The inputs of the controller are the set speed v_{set}, the desired time gap T_{gap}, the ego velocity v_{ego}, the distance $D_{\text{rel}} = x_{\text{lead}} - x_{\text{ego}}$, as well as the relative velocity v_{rel}, and the output is $a_{\text{c,ego}}$.

[2] https://github.com/verivital/ARCH-COMP2022

Specifications The verification objective of this system is that given a scenario where both cars are driving safely, the lead car suddenly slows down with $a_{c,\text{lead}}$ = -2. We want to check whether there is a collision in the following 5 s. Formally, this safety specification of the system can be expressed as $D_{\text{rel}} \geq D_{\text{safe}}$, where $D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \cdot v_{\text{ego}}$, and $T_{\text{gap}} = 1.4$ s and $D_{\text{default}} = 10$. The initial conditions are: $x_{\text{lead}}(0) \in$ [90,110], $v_{\text{lead}}(0) \in$ [32,32.2], $a_{\text{lead}}(0) = a_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.2], $x_{\text{ego}} \in$ [10,11]. A control period of 0.1 s is used.

3.2 Sherlock-Benchmark-9 (TORA)

This benchmark considers translational oscillations by a rotational actuator (TORA) [19, 28], where a cart is attached to a wall with a spring and is free to move on a friction-less surface. The cart itself has a weight attached to an arm inside it, which is free to rotate about an axis. This serves as the control input, in order to stabilize the cart at $\mathbf{x} = \mathbf{0}$. The model is a four-dimensional system, given by the following equations [28, eq. (4)]:

$$\dot{x_1} = x_2, \quad \dot{x_2} = -x_1 + 0.1\sin(x_3), \quad \dot{x_3} = x_4, \quad \dot{x_4} = u. \tag{2}$$

There are three neural network controllers for this benchmark: the first one has three ReLU hidden layers and a linear output layer. This controller was trained using a data-driven model predictive controller proposed in [20]. Note that the output of the neural network $f(\mathbf{x})$ needs to be normalized in order to obtain u, namely $u = f(\mathbf{x}) - 10$. The sampling time for this controller is 1 s and we verify it against specification 1 below. The other two controllers have three hidden layers of 20 neurons each, and one output layer. In contrast to the first controller, we use sigmoid activation functions for the hidden layers and a tanh output layer. The sampling time of these controllers is 0.5 s, the output of the neural network $f(\mathbf{x})$ needs to be postprocessed as $u = 11 \cdot f(\mathbf{x})$, and we verify them against specification 2 below.

Specification 1. This is a safety specification. For an initial set of $x_1 \in [0.6, 0.7]$, $x_2 \in [-0.7, -0.6]$, $x_3 \in [-0.4, -0.3]$, and $x_4 \in [0.5, 0.6]$, the system states have to stay within the box $\mathbf{x} \in [-2, 2]^4$ for a time window of 20 s.

Specification 2. For an initial set of $x_1 \in$ [-0.77, -0.75], $x_2 \in$ [-0.45, -0.43], $x_3 \in$ [0.51, 0.54], and $x_4 \in [-0.3, -0.28]$, it is required that the system reaches the set $x_1 \in [-0.1, 0.2]$, $x_2 \in [-0.9, -0.6]$ within a time window of 5 s.

3.3 Sherlock-Benchmark-10 (Unicycle Car Model)

This benchmark considers a unicycle model of a car [19] with the x and y coordinates on a two-dimensional plane, the velocity magnitude (speed), and steering angle as state variables. The dynamic equations are (see [5, Sec. III.B]; a different input is used here):

$$\dot{x}_1 = x_4\cos(x_3), \quad \dot{x}_2 = x_4\sin(x_3), \quad \dot{x}_3 = u_2, \quad \dot{x}_4 = u_1 + w, \tag{3}$$

where w is a bounded error in the range $10^{-4}[-1, 1]$. A neural network controller was trained for this system using a model predictive controller as a "demonstrator" or "teacher". The trained network has one hidden layer with 500 neurons. Note that the output of the neural network $f(\mathbf{x})$ needs to be normalized in order to obtain (u_1, u_2), namely $u_i = f(\mathbf{x})_i - 20$. The sampling time for this controller is 0.2 s.

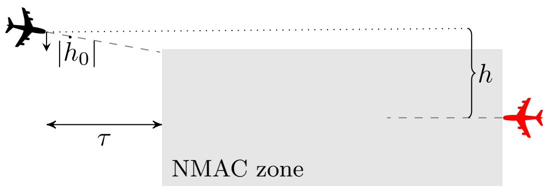


Figure 2: VerticalCAS encounter geometry

Specification This is a reachability specification. For an initial set of $x_1 \in [9.5, 9.55]$, $x_2 \in [-4.5, -4.45]$, $x_3 \in [2.1, 2.11]$, and $x_4 \in [1.5, 1.51]$, the system has to reach the set $x_1 \in [-0.6, 0.6], x_2 \in [-0.2, 0.2], x_3 \in [-0.06, 0.06], x_4 \in [-0.3, 0.3]$ within a time window of 10 s.

3.4 VCAS Benchmark

This benchmark is a closed-loop variant of the *aircraft collision avoidance system ACAS X*. The scenario involves two aircraft, the ownship and the intruder, where the ownship is equipped with a collision avoidance system referred to as VerticalCAS [32]. Every second, VerticalCAS issues vertical climb rate advisories to the ownship pilot to avoid a near mid-air collision (NMAC). Near mid-air collisions are regions in which the ownship and the intruder are separated by less than 100ft vertically and 500ft horizontally. The ownship (black) is assumed to have a constant horizontal speed, and the intruder (red) is assumed to follow a constant horizontal trajectory towards ownship, see Figure 2. The current geometry of the system is described by

- h, intruder altitude relative to ownship,
- $\dot{h}_0$, ownship vertical climb rate, and
- τ, the time until the ownship (black) and intruder (red) are no longer horizontally separated.

We can, therefore, assume that the intruder is static and the horizontal separation τ decreases by one each second. There are nine advisories and each of them instructs the pilot to accelerate until the vertical climb rate of the ownship complies with the advisory:

1. COC: Clear Of Conflict;
2. DNC: Do Not Climb;
3. DND: Do Not Descend;
4. DES1500: Descend at least 1500 ft/s;
5. CL1500: Climb at least 1500 ft/s;
6. SDES1500: Strengthen Descent to at least 1500 ft/s;
7. SCL1500: Strengthen Climb to at least 1500 ft/s;
8. SDES2500: Strengthen Descent to at least 2500 ft/s;
9. SCL2500: Strengthen Climb to at least 2500 ft/s.

In addition to the parameters describing the geometry of the encounter, the current state of the system stores the advisory $\text{adv} \in \{1, \ldots, 9\}$ (numbers correspond to the above list) issued to the ownship at the previous time step. VerticalCAS is implemented as nine ReLU networks N_i, one for each (previous) advisory, with three inputs $(h, \dot{h}_0, \tau)$, five fully-connected hidden layers of 20 units each, and nine outputs representing the score of each possible advisory. Therefore, given a current state $(h, \dot{h}_0, \tau, \text{adv})$, the new advisory adv' is obtained by computing the argmax of the output of N_{adv} on $(h, \dot{h}_0, \tau)$.

Given the new advisory, the pilot can choose acceleration $\ddot{h}_0$ as follows. If the new advisory is COC, then it can be any acceleration from the set $\{-\frac{g}{8}, 0, \frac{g}{8}\}$. For all remaining advisories, if the previous advisory coincides with the new one and the current climb rate complies with the new advisory (e.g., $\dot{h}_0$ is non-positive for DNC and $\dot{h}_0 \geq 1500$ for CL1500), the acceleration $\ddot{h}_0$ is 0; otherwise, the pilot can choose any acceleration $\ddot{h}_0$ from the given sets:

- DNC: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$;
- DND: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$;
- DES1500: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$;
- CL1500: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$;
- SDES1500: $\{-\frac{g}{3}\}$;
- SCL1500: $\{\frac{g}{3}\}$;
- SDES2500: $\{-\frac{g}{3}\}$;
- SCL2500: $\{\frac{g}{3}\}$,

where g represents the gravitational constant 32.2 ft/s^2.

It was proposed to tweak the benchmark for the tools that cannot account for all possible choices of acceleration efficiently. Those tools can consider two strategies for picking a single acceleration at each time step:

- a worst-case scenario selection, where we choose the acceleration that will take the ownship closer to or less far apart from the intruder.
- always select the acceleration in the middle.

Given the current system state $(h, \dot{h}_0, \tau, \text{adv})$, the new advisory adv' and the acceleration $\ddot{h}_0$, the new state of the system can be computed by the following equations [32, eq. (15)]:

$$\begin{aligned} h(k+1) &= h(k) - \dot{h}_0(k)\Delta\tau - 0.5\ddot{h}_0(k)\Delta\tau^2 \\ \dot{h}_0(k+1) &= \dot{h}_0(k) + \ddot{h}_0(k)\Delta\tau \\ \tau(k+1) &= \tau(k) + \Delta\tau \\ \text{adv}(k+1) &= \text{adv}' \end{aligned}$$

where $\Delta\tau = 1$.

Specification The ownship has to be outside of the NMAC zone after $k \in \{1, \ldots, 10\}$ time steps, i.e., $h(k) > 100$ or $h(k) < -100$, for all possible choices of acceleration by the pilot. The set of initial states considered is: $h(0) \in [-133, -129]$, $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, $\tau(0) = 25$ and $\text{adv}(0) = \text{COC}$.

3.5 Single-Pendulum Benchmark

We consider a classical inverted pendulum. A ball of mass m is attached to a massless beam of length L. The beam is actuated with a torque T and we assume viscous friction with a friction coefficient of c. The governing equation of motion can be obtained as [39, eq. (1)]:

$$\ddot{\theta} = \frac{g}{L}\sin\theta + \frac{1}{mL^2}\left(T - c\,\dot{\theta}\right), \tag{4}$$

where θ is the angle of the link with respect to the upward vertical axis and $\dot{\theta}$ is the angular velocity. After defining the state variables $x_1 = \theta$ and $x_2 = \dot{\theta}$, the dynamics in state-space form is

$$\dot{x}_1 = x_2 \tag{5a}$$

$$\dot{x}_2 = \frac{g}{L}\sin x_1 + \frac{1}{mL^2}\left(T - c\,x_2\right) \tag{5b}$$

Controllers are trained using *behavior cloning*, a supervised learning approach for training controllers. The code as well as training procedures are provided. The model parameters are chosen as

$$m = 0.5, L = 0.5, c = 0, g = 1 \tag{6}$$

and the time step for the controller and the discrete-time model is $\Delta t = 0.05$. The initial set is

$$x \in [1.0, 1.2] \times [0.0, 0.2].$$

Specification $\forall t \in [0.5, 1] : \theta \in [0, 1]$ (analogously for $k \in [10, 20]$ in discrete time).

3.6 Double-Pendulum Benchmark

The double pendulum is an inverted two-link pendulum with equal point masses m at the end of connected mass-less links of length L. The links are actuated with torques T_1 and T_2 and we assume viscous friction exists with a coefficient of c. The governing equations of motion are described by the following equations [39, eq. (3a-b)]:

$$2\ddot{\theta}_1 + \ddot{\theta}_2\cos(\theta_2 - \theta_1) - \dot{\theta}_2^2\sin(\theta_2 - \theta_1) - 2\frac{g}{L}\sin\theta_1 + \frac{c}{mL^2}\dot{\theta}_1 = \frac{1}{mL^2}T_1 \tag{7a}$$

$$\ddot{\theta}_1\cos(\theta_2 - \theta_1) + \ddot{\theta}_2 + \dot{\theta}_1^2\sin(\theta_2 - \theta_1) - \frac{g}{L}\sin\theta_2 + \frac{c}{mL^2}\dot{\theta}_2 = \frac{1}{mL^2}T_2 \tag{7b}$$

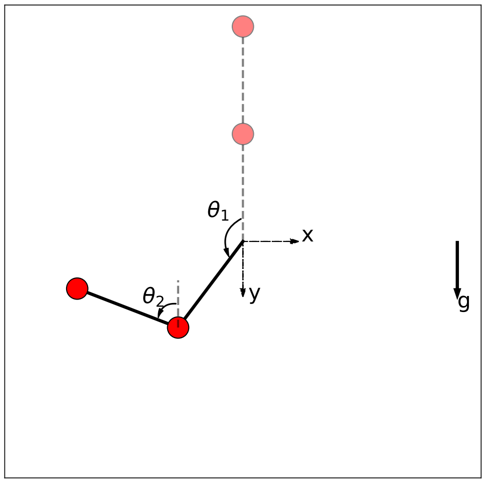


Figure 3: Inverted double pendulum. The goal is to keep the pendulum upright (dashed schematics)

where θ_1 and θ_2 are the angles of the links with respect to the upward vertical axis (see Figure 3) and g is the gravitational acceleration. After defining the state vector as $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$, the dynamics in state-space form is

$$\dot{x}_1 = x_3 \tag{8a}$$

$$\dot{x}_2 = x_4 \tag{8b}$$

$$\dot{x}_3 = \frac{1}{2\left(\frac{\cos^2(x_1-x_2)}{2} - 1\right)} \cos(x_1 - x_2) \left({x_3}^2 \sin(x_1 - x_2) - \cos(x_1 - x_2) \left(\frac{g \sin(x_1)}{L} \right.\right. \tag{8c}$$

$$\left.\left. - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{T_1 - c\, x_3}{2\, L^2\, m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\, x_4}{L^2\, m} \right) \tag{8d}$$

$$- \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{g \sin(x_1)}{L} + \frac{T_1 - c\, x_3}{2\, L^2\, m} \tag{8e}$$

$$\dot{x}_4 = \frac{-1}{\frac{\cos^2(x_1-x_2)}{2} - 1} \left({x_3}^2 \sin(x_1 - x_2) - \cos(x_1 - x_2) \left(\frac{g \sin(x_1)}{L} - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} \right.\right. \tag{8f}$$

$$\left.\left. + \frac{T_1 - c\, x_3}{2\, L^2\, m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\, x_4}{L^2\, m} \right), \tag{8g}$$

The controllers for the double pendulum benchmark are obtained using the same methods as the controllers for the single pendulum benchmark; also here, the code as well as training procedures are provided. The model parameters are chosen as in (6) The initial set is

$$x \in [1.0, 1.3]^4.$$

Specification 1 $\forall t \in [0, 1] : x \in [-1.0, 1.7]^4$ (analogously for $k \in [0, 20]$ in discrete time) for $\Delta t = 0.05$.

Specification 2 $\forall t \in [0, 0.4] : x \in [-0.5, 1.5]^4$ (analogously for $k \in [0, 20]$ in discrete time) for $\Delta t = 0.02$.

We verify *controller double pendulum less robust* against specification 1 and *controller double pendulum more robust* against specification 2.

3.7 Airplane Benchmark

The airplane example consists of a dynamical system that is a simple model of a flying airplane as shown in Figure 4. The state is

$$x = [s_x, s_y, s_z, v_x, v_y, v_z, \phi, \theta, \psi, r, p, q]^T, \tag{9}$$

where (s_x, s_y, s_z) is the position of the center of gravity, (v_x, v_y, v_z) are the components of velocity in (x, y, z) directions, (p, q, r) are body rotation rates, and (ϕ, θ, ψ) are the Euler angles. The equations of motion are reduced to [39, eq. (7)]:

$$\dot{v}_x = -g\sin\theta + \frac{F_x}{m} - qv_z + rv_y \tag{10a}$$

$$\dot{v}_y = g\cos\theta\sin\phi + \frac{F_y}{m} - rv_x + pv_z \tag{10b}$$

$$\dot{v}_z = g\cos\theta\cos\phi + \frac{F_z}{m} - pv_y + qv_x \tag{10c}$$

$$I_x\dot{p} + I_{xz}\dot{r} = M_x - (I_z - I_y)qr - I_{xz}pq \tag{10d}$$

$$I_y\dot{q} = M_y - I_{xz}\left(r^2 - p^2\right) - (I_x - I_z)pr \tag{10e}$$

$$I_{xz}\dot{p} + I_z\dot{r} = M_z - (I_y - I_x)qp - I_{xz}rq. \tag{10f}$$

The mass of the airplane is denoted with m and I_x, I_y, I_z and I_{xz} are the moment of inertia with respect to the indicated axis; see Figure 4. The control parameters include three force components F_x, F_y and F_z and three moment components M_x, M_y, M_z. Note that for simplicity, we assume that the aerodynamic forces are absorbed in the force vector F. In addition to these six equations, we have six additional kinematic equations [39, eq. (8,9)]:

$$\begin{bmatrix}\dot{s}_x\\ \dot{s}_y\\ \dot{s}_z\end{bmatrix} = \begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix} \tag{11}$$

and

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}1 & \tan\theta\sin\phi & \tan\theta\cos\phi\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix}. \tag{12}$$

As in the pendulum benchmarks, controllers are trained for the airplane problem using behavior cloning. The system involves the model parameters

$$m = 1, I_x = I_y = I_z = 1, I_{xz} = 0, g = 1$$

and the time step for the controller and the discrete-time model is $\Delta t = 0.1$. The initial set is

$$x = y = z = r = p = q = 0, \quad [v_x, v_y, v_z, \phi, \theta, \psi] \in [0.0, 1.0]^6.$$

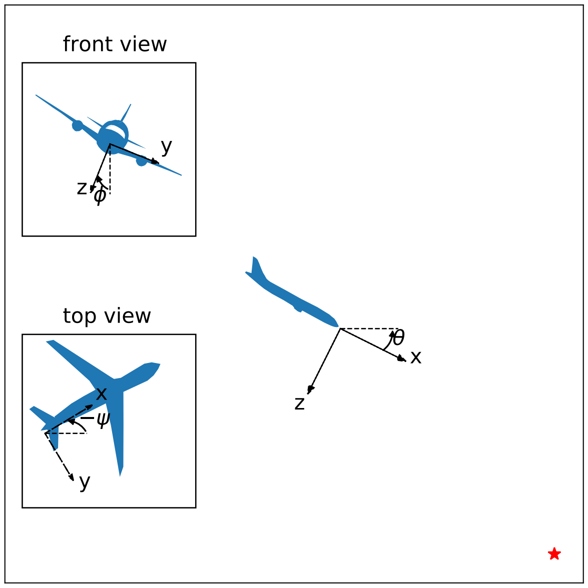


Figure 4: The airplane example.

Specification $\forall t \in [0, 2] : s_y \in [-0.5, 0.5]$, $[\phi, \theta, \psi] \in [-1.0, 1.0]^3$ (analogously for $k \in [0, 20]$ in discrete time).

We verify *controller airplane* against the above specification.

3.8 Benchmark: Attitude Control

We consider the attitude control of a rigid body with six states and three inputs [43, 48]. The system dynamics is given by [43, Sec. V]:

$$\begin{aligned}
&\dot{\omega}_1 = 0.25(u_0 + \omega_2\omega_3), \qquad \dot{\omega}_2 = 0.5(u_1 - 3\omega_1\omega_3), \qquad \dot{\omega}_3 = u_2 + 2\omega_1\omega_2,\\
&\dot{\psi}_1 = 0.5\left(\omega_2(\psi_1^2+\psi_2^2+\psi_3^2-\psi_3)+\omega_3(\psi_1^2+\psi_2^2+\psi_2+\psi_3^2)+\omega_1(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),\\
&\dot{\psi}_2 = 0.5\left(\omega_1(\psi_1^2+\psi_2^2+\psi_3^2+\psi_3)+\omega_3(\psi_1^2-\psi_1+\psi_2^2+\psi_3^2)+\omega_2(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),\\
&\dot{\psi}_3 = 0.5\left(\omega_1(\psi_1^2+\psi_2^2-\psi_2+\psi_3^2)+\omega_2(\psi_1^2+\psi_1+\psi_2^2+\psi_3^2)+\omega_3(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),
\end{aligned}$$

wherein the state $x = (\omega^T, \psi^T)^T$ consists of the angular velocity vector in a body-fixed frame $\omega \in \mathbb{R}^3$ and the Rodrigues parameter vector $\psi \in \mathbb{R}^3$.

The control torque $u \in \mathbb{R}^3$ is updated every 0.1 s by a neural network with three hidden layers, each of which has 64 neurons. The activations of the hidden layers are sigmoid and identity, respectively. We train the neural-network controller using supervised learning methods to learn from a known nonlinear controller [43]. The initial state set is:

$$\begin{aligned}
&\omega_1 \in [-0.45, -0.44], \omega_2 \in [-0.55, -0.54], \omega_3 \in [0.65, 0.66],\\
&\psi_1 \in [-0.75, -0.74], \psi_2 \in [0.85, 0.86], \psi_3 \in [-0.65, -0.64].
\end{aligned}$$

Specification The system should not reach the following unsafe set in 3 s (30 time steps):

$$\omega_1 \in [-0.2, 0], \omega_2 \in [-0.5, -0.4], \omega_3 \in [0, 0.2],$$
$$\psi_1 \in [-0.7, -0.6], \psi_2 \in [0.7, 0.8], \psi_3 \in [-0.4, -0.2].$$

We would like to show that the above specification does not hold.

3.9 Benchmark: QUAD

This benchmark studies a neural-network controlled quadrotor (QUAD) with twelve state variables [10]. We have the inertial (north) position x_1, the inertial (east) position x_2, the altitude x_3, the longitudinal velocity x_4, the lateral velocity x_5, the vertical velocity x_6, the roll angle x_7, the pitch angle x_8, the yaw angle x_9, the roll rate x_{10}, the pitch rate x_{11}, and the yaw rate x_{12}. The control torque $u \in \mathbb{R}^3$ is updated every 0.1 s by a neural network with 3 hidden layers, each of which has 64 neurons. The activations of the hidden layers and the output layer are sigmoid and identity, respectively. The dynamics are given by the following equations [10, eq. (12-16)]:

$$\begin{aligned}
\dot{x}_1 =& \cos(x_8)\cos(x_9)x_4 + \left(\sin(x_7)\sin(x_8)\cos(x_9) - \cos(x_7)\sin(x_9)\right)x_5 \\
& + \left(\cos(x_7)\sin(x_8)\cos(x_9) + \sin(x_7)\sin(x_9)\right)x_6 \\
\dot{x}_2 =& \cos(x_8)\sin(x_9)x_4 + \left(\sin(x_7)\sin(x_8)\sin(x_9) + \cos(x_7)\cos(x_9)\right)x_5 \\
& + \left(\cos(x_7)\sin(x_8)\sin(x_9) - \sin(x_7)\cos(x_9)\right)x_6 \\
\dot{x}_3 =& \sin(x_8)x_4 - \sin(x_7)\cos(x_8)x_5 - \cos(x_7)\cos(x_8)x_6 \\
\dot{x}_4 =& x_{12}x_5 - x_{11}x_6 - g\sin(x_8) \\
\dot{x}_5 =& x_{10}x_6 - x_{12}x_4 + g\cos(x_8)\sin(x_7) \\
\dot{x}_6 =& x_{11}x_4 - x_{10}x_5 + g\cos(x_8)\cos(x_7) - g - u_1/m \\
\dot{x}_7 =& x_{10} + \sin(x_7)\tan(x_8)x_{11} + \cos(x_7)\tan(x_8)x_{12} \\
\dot{x}_8 =& \cos(x_7)x_{11} - \sin(x_7)x_{12} \\
\dot{x}_9 =& \frac{\sin(x_7)}{\cos(x_8)}x_{11} - \frac{\cos(x_7)}{\cos(x_8)}x_{12} \\
\dot{x}_{10} =& \frac{J_y - J_z}{J_x}x_{11}x_{12} + \frac{1}{J_x}u_2 \\
\dot{x}_{11} =& \frac{J_z - J_x}{J_y}x_{10}x_{12} + \frac{1}{J_y}u_3 \\
\dot{x}_{12} =& \frac{J_x - J_y}{J_z}x_{10}x_{11} + \frac{1}{J_z}\tau_\psi
\end{aligned}$$

where

$$g = 9.81, \quad m = 1.4, \quad J_x = 0.054,$$
$$J_y = 0.054, \quad J_z = 0.104, \quad \tau_\psi = 0.$$

The initial set is:

$$x_1 \in [-0.4, 0.4], x_2 \in [-0.4, 0.4], x_3 \in [-0.4, 0.4], x_4 \in [-0.4, 0.4],$$
$$x_5 \in [-0.4, 0.4], x_6 \in [-0.4, 0.4], x_7 = 0, x_8 = 0, x_9 = 0, x_{10} = 0, x_{11} = 0, x_{12} = 0.$$

Specification The control goal is to stabilize the attitude x_3 to a goal region $[0.94, 1.06]$ and remain within these bounds with a time horizon of 5 s (50 time steps).

3.10 2D Spacecraft Docking

In the 2D spacecraft docking environment, the state of an active deputy spacecraft is expressed relative to the passive chief spacecraft in Hill's reference frame [25]. The dynamics are given by a first-order approximation of the relative motion dynamics between the deputy and chief spacecraft, which is given by Clohessy-Wiltshire [18] equations [45, eq. (12)],

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_y \\ \ddot{s}_x \\ \ddot{s}_y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 2n \\ 0 & 0 & -2n & 0 \end{bmatrix} \begin{bmatrix} s_x \\ s_y \\ \dot{s}_x \\ \dot{s}_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & \frac{1}{m} \end{bmatrix} u, \tag{13}$$

where $m = 12$ (kg), $n = 0.001027$ (rad/s), and $u \in \mathbb{R}^2$.

The neural network controller was trained on the Docking 2D environment with reinforcement learning using the training procedure described in [45]. However, the training procedure differed in providing only the full state (position and velocity) as input and with hard clipping of output actions replaced with soft tanh clipping. The neural network architecture was a shallow multilayer perceptron with 2 hidden layers of 256 neurons and tanh activation functions, and a linear output layer. The pre-processing and post-processing of the controller has been incorporated into the model as linear layers. The controller was trained with a sampling time of 1 s.

Specification The spacecraft should satisfy the following safety constraints for 40 s:

$$(\dot{s}_x^2 + \dot{s}_y^2)^{\frac{1}{2}} \leq 0.2 + 2n(s_x^2 + s_y^2)^{\frac{1}{2}}, \tag{14}$$

given the initial set is

$$s_x \in [70, 106], s_y \in [70, 106], \dot{s}_x \in [-0.28, 0.28], \dot{s}_y \in [-0.28, 0.28].$$

4 Verification Results

For each of the participating tools, we obtained verification results for each of the proposed benchmarks. Reachable sets are shown for those methods that are able to construct them.

4.1 JuliaReach

This subsection presents the results of *JuliaReach*. JuliaReach was able to analyze eight benchmark problems (four verified, four falsified). For each problem, JuliaReach uses slightly different settings as described below.

4.1.1 ACC

Using the parameters `abstol=1e-6, orderT=6, orderQ=1`, JuliaReach verifies the property $D_{\text{rel}} \geq D_{\text{safe}}$ for the whole time horizon in less than a second. The reach sets of D_{rel} and D_{safe} together with some simulations are shown in Figure 5.

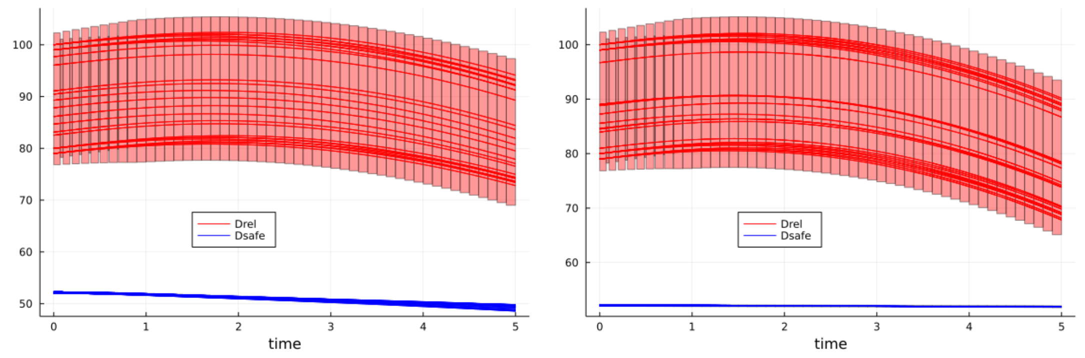


Figure 5: ***JuliaReach.*** Analysis results for the ACC benchmark and the ReLU controller (left) resp. the tanh controller (right). The plot additionally shows simulations.

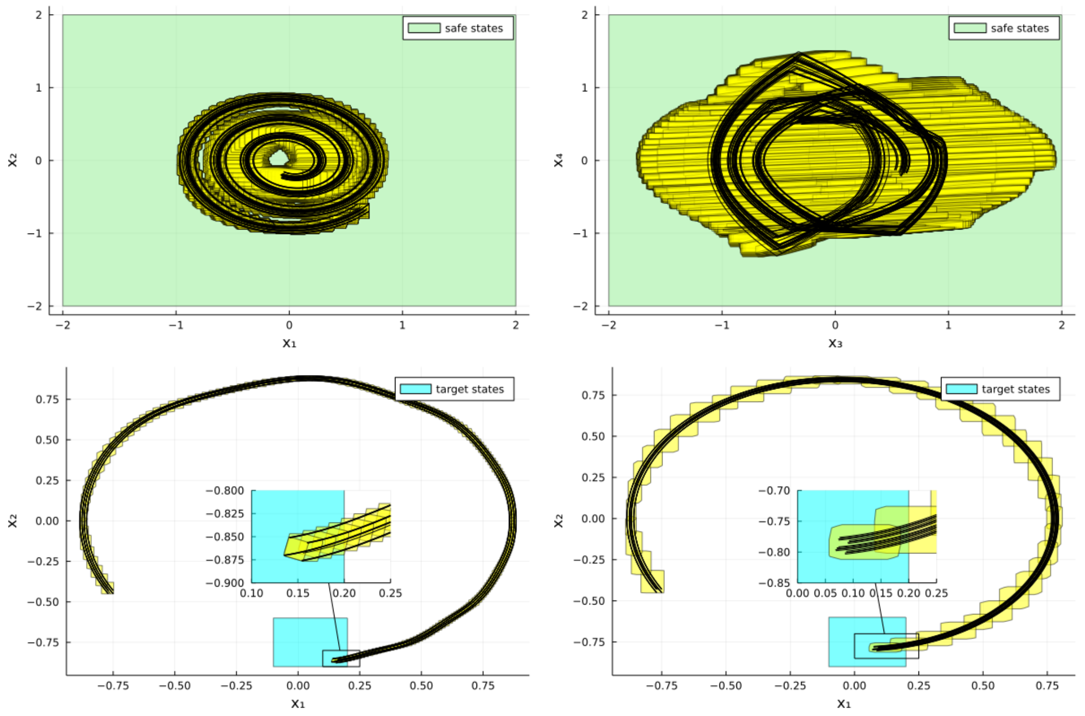


Figure 6: ***JuliaReach.*** Analysis results for the TORA benchmark for the ReLU controller (top), the sigmoid controller (bottom left), resp. the ReLU/tanh controller (bottom right). The plots additionally show simulations.

4.1.2 Sherlock-Benchmark-9

We analyze three different controllers for the TORA benchmark problem. For the ReLU controller, the approximation error is hard to tame for the JuliaReach approach. To maintain enough precision to prove correctness, the initial states are split into $4 \times 4 \times 3 \times 5$ boxes. While each box spawns an independent analysis that could be parallelized, the run time of

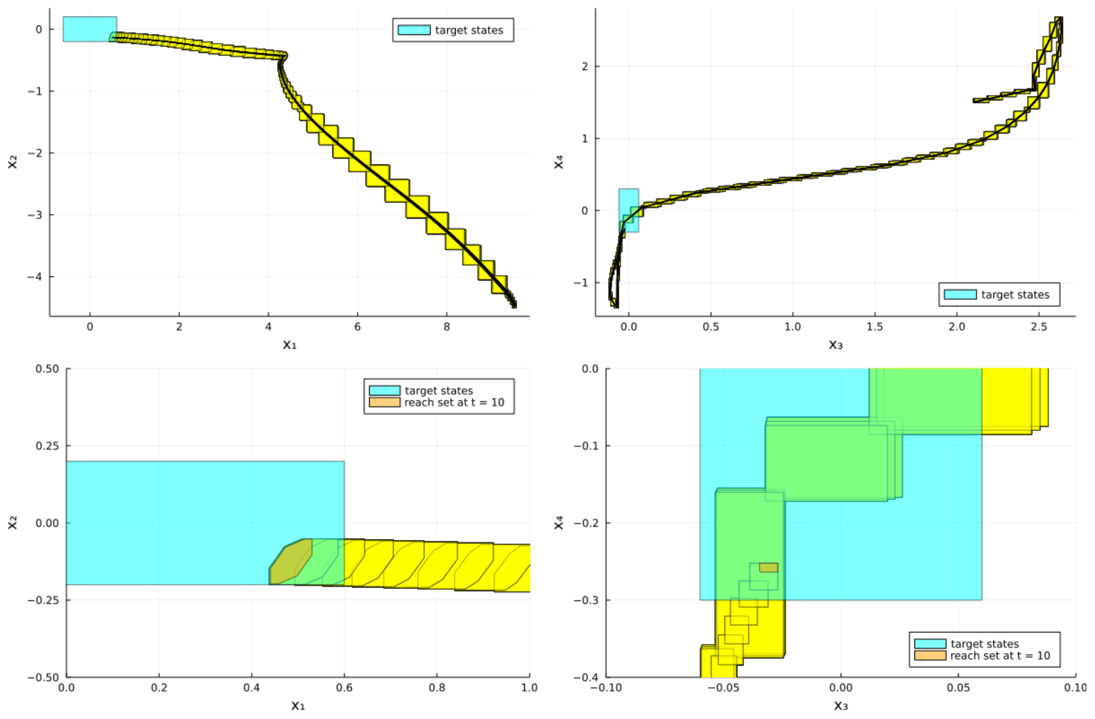


Figure 7: ***JuliaReach.*** Analysis results for the Unicycle benchmark. The first two plots show the overall reach sets and simulations. The other two plots show a close-up of the target set. The orange subset of the last reach set is obtained at time point $t = 10$.

the sequential execution is 34 minutes. We use the parameters `abstol=1e-10, orderT=8, orderQ=3`. The reach sets of all 240 runs together with some simulations, projected to x_1/x_2 resp. x_3/x_4, are shown in Figure 6. The sigmoid and ReLU/tanh controllers reach the target set within the given time constraints, as shown in the x_1/x_2 projections in Figure 6. The latter analyses take 7 s respectively 2 s.

4.1.3 Sherlock-Benchmark-10

The disturbance w is modeled here as a constant with uncertain initial value. Simulations show that the target set is reached only in the last moment, so the analysis requires a high precision to prove containment of the last reach set. Using the parameters `abstol=1e-15, orderT=10, orderQ=1` and splitting the initial states into $3 \times 1 \times 8 \times 1$ boxes, JuliaReach verifies the property in 93 s. The reach sets of all 24 runs together with some simulations, projected to x_1/x_2 resp. x_3/x_4, are shown in Figure 7. JuliaReach can evaluate the Taylor polynomial at the time point $t = 10$ (rather than the last time *interval*), which results in a more precise result (as shown in the plots), although additional precision is not required for this problem, as the reach set for the last time interval is already fully contained in the target set.

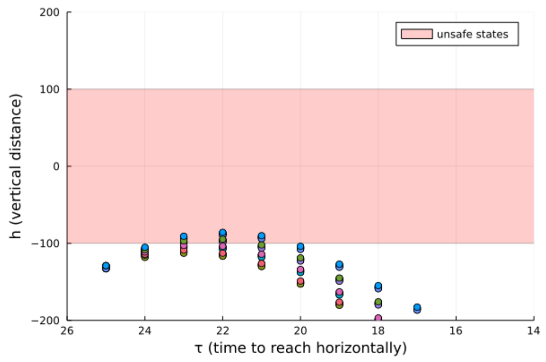


Figure 8: ***JuliaReach.*** Simulations of the VCAS benchmark.

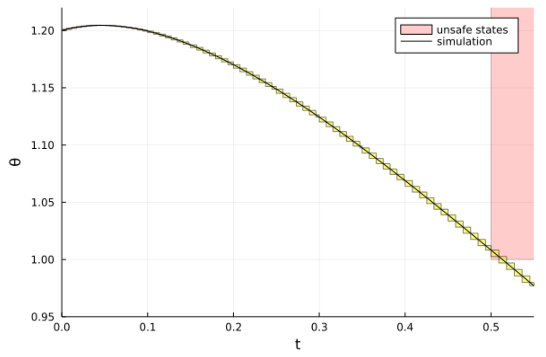


Figure 9: ***JuliaReach.*** Analysis results for the Single-Pendulum benchmark until time $t = 0.55$. The plot additionally shows a simulation.

4.1.4 VCAS

The VCAS benchmark problem differs from the other problems in that it uses multiple controllers and discrete time. There is currently no native support for this setting in JuliaReach, so a custom simulation algorithm that always chooses the central acceleration was used. JuliaReach produces ten simulations in 1 s, which indicate satisfaction for the initial values $\dot{h}(0) \in \{-19.5, -22.5\}$ but show violation of the property for the other initial values. The simulation results are shown in Figure 8.

4.1.5 Single Pendulum

This system violates the specification; hence it suffices to start the analysis from a subset of the initial states and interrupt when a violation is detected. Here, starting from the highest coordinate in each dimension, a violation occurs within eleven control periods. Using the parameters `abstol=1e-7, orderT=4, orderQ=1`, JuliaReach falsifies the property in 0.5 s. Figure 9 shows the reach sets together with a simulation projected to time and θ.

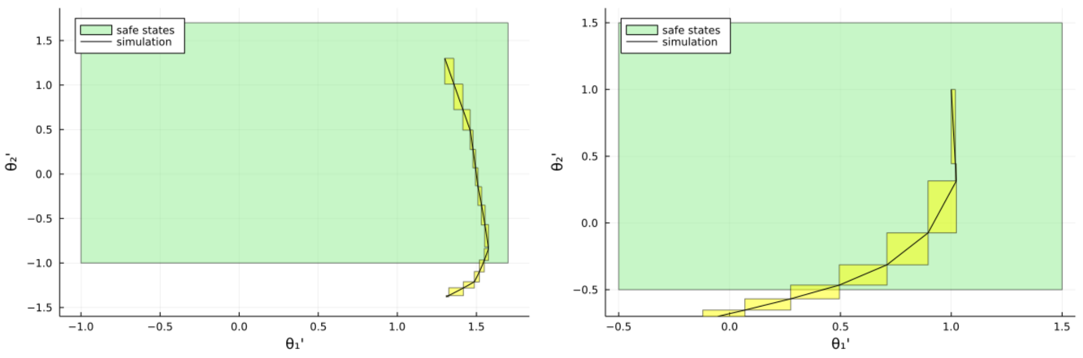


Figure 10: ***JuliaReach.*** Analysis results for the double-pendulum benchmark, including a simulation. The first plot shows the results for the less robust controller until time $t = 0.25$. The second plot shows the results for the more robust controller until time $t = 0.14$.

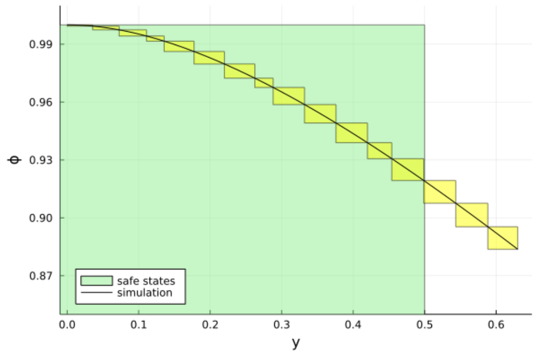


Figure 11: ***JuliaReach.*** Analysis results for the airplane benchmark until time $t = 0.4$, including a simulation.

4.1.6 Double Pendulum

Similarly to the single-pendulum problem, this system violates the specification for both controllers; hence it suffices to start the analysis from a subset of the initial states and interrupt when a violation is detected. Considering the less robust controller, when starting from the highest value in each dimension, a violation occurs within five control periods. Similarly, considering the more robust controller and starting from the lowest value in each dimension, a violation occurs within seven control periods. Using the parameters `abstol=1e-9, orderT=8, orderQ=1` and an older version of the Taylor-model algorithm, JuliaReach falsifies the property in 11 s (less robust controller) resp. 2 s (more robust controller). Figure 10 shows the reach sets together with a simulation projected to $\dot{\theta}_1/\dot{\theta}_2$.

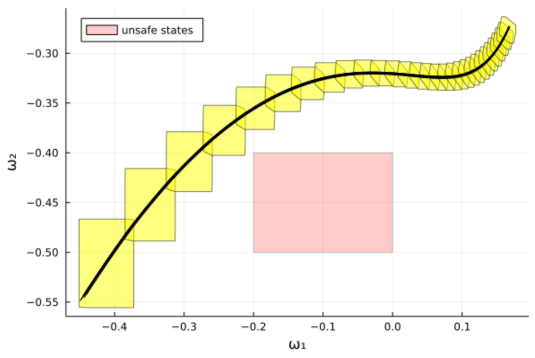


Figure 12: ***JuliaReach.*** Analysis results for the attitude control benchmark, including simulations.

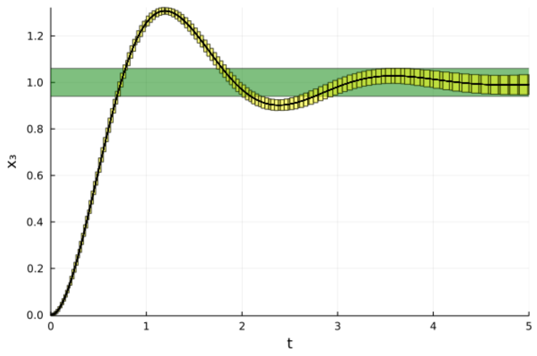


Figure 13: ***JuliaReach.*** Analysis results for the quadrotor benchmark, including simulations.

4.1.7 Airplane

This system violates the specification; hence it suffices to start the analysis from a subset of the initial states and interrupt when a violation is detected. When starting from the highest coordinate in each dimension, a violation occurs immediately in dimension θ and within four control periods in dimension y. To obtain some nontrivial results, JuliaReach ignores the violation in dimension θ. Using the parameters `abstol=1e-10, orderT=7, orderQ=1`, JuliaReach falsifies the property in 9 s. The reach sets together with a simulation, projected to y/ϕ, are shown in Figure 11.

4.1.8 Attitude Control

Using the parameters `abstol=1e-6, orderT=6, orderQ=1`, JuliaReach verifies the property in 3 s. The reach sets together with some simulations are shown in Figure 12.

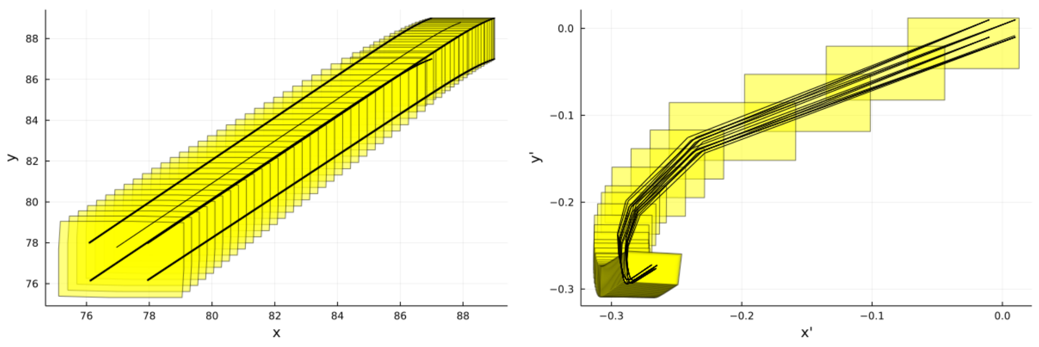


Figure 14: ***JuliaReach.*** Analysis results for the Spacecraft benchmark, including simulations.

4.1.9 Quadrotor

Although simulations indicate that this controller is safe, the precision of JuliaReach is not high enough to prove it. The property can be proven for a smaller initial set $[-0.004, 0.004]^6 \times \{0\}^6$. Using the parameters `abstol=1e-8, orderT=5, orderQ=1`, JuliaReach verifies the property in 14 s. The reach sets together with some simulations are shown in Figure 13.

4.1.10 Spacecraft

Although simulations indicate that this controller is safe, the precision of JuliaReach is not high enough to prove it. The property can be proven for a smaller initial set $[87, 89]^2 \times [-0.01, 0.01]^2$. Using the parameters `abstol=1e-10, orderT=5, orderQ=1`, JuliaReach verifies the property in 1 s. The reach sets together with some simulations are shown in Figure 14. Since the property is four-dimensional, it cannot be illustrated in the plot.

4.2 NNV

We present the results utilizing *NNV* on each of the benchmarks. We have been able to encode all the benchmarks into NNV and attempted to verify all of them, however, due to the conservativeness of our methods and the complexity of the different benchmarks, we were only able to verify or falsify the ACC, TORA with one out of the 3 controllers, VCAS, airplane, single pendulum, and double pendulum. NNV was unable to verify the unicycle, quadrotor, attitude and docking spacecraft benchmarks.

4.2.1 ACC

NNV is able to verify the safety property $D_{\text{rel}} \geq D_{\text{safe}}$ for the whole time horizon in 18.6 s. The reach sets of D_{rel} and D_{safe} are shown in Figure 15.

4.2.2 Tora

NNV is able to verify one of the three controllers for the tora benchmark, the relu/tanh controller in 1 hour and 17 minutes, by partitioning the initial state into 4×8×6×4 boxes. The reach sets are shown in Figure 16.

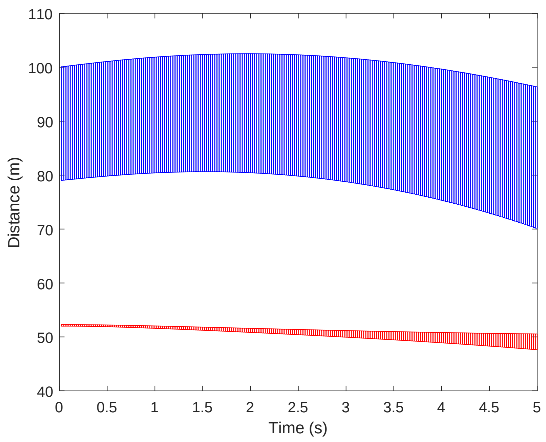
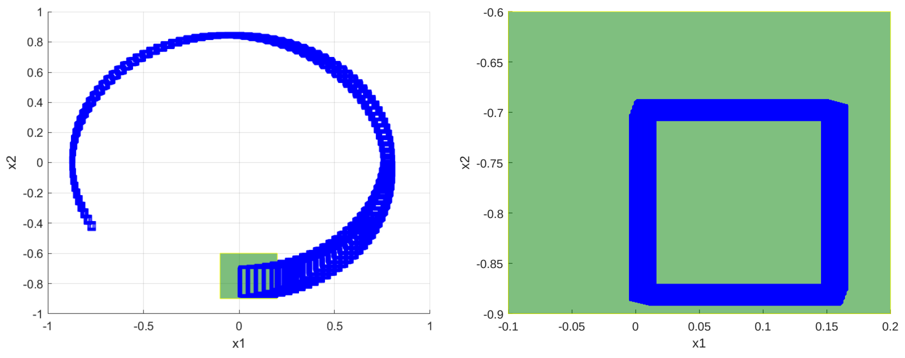


Figure 15: ***NNV.*** Analysis results for the ACC benchmark showing the sets of $D_{\mathbf{rel}}$ and $D_{\mathbf{safe}}$.



(a) Trajectory (b) Last

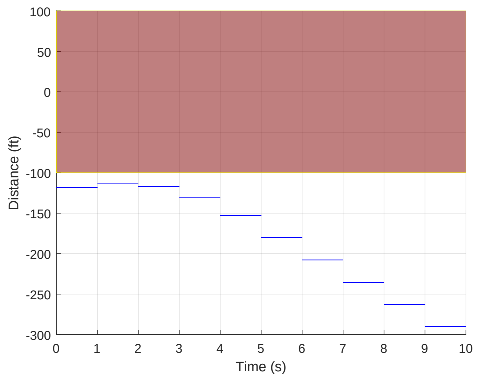
Figure 16: ***NNV.*** Analysis results for the Tora benchmark showing the tora sets in **blue** and the goal region in **green**, when using the controller with relu hidden layers and tanh output layer.

4.2.3 VCAS

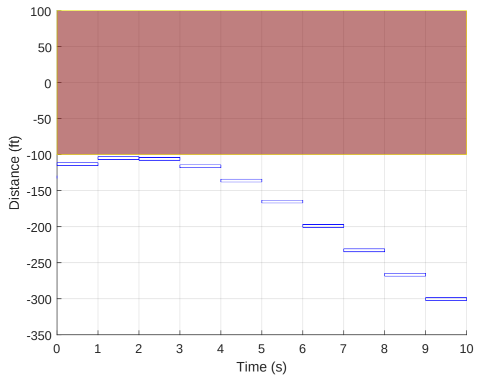
NNV is able to verify the NMAC safety property for the whole time horizon for each of the cases. There are 5 cases where we prove that the system is unsafe, and 3 where the system is safe, which corresponds to [middle, 19.5], [middle, 22.5] and [worst, 19.5]. These results are depicted in Figures 17 and 18.

4.2.4 Single Pendulum

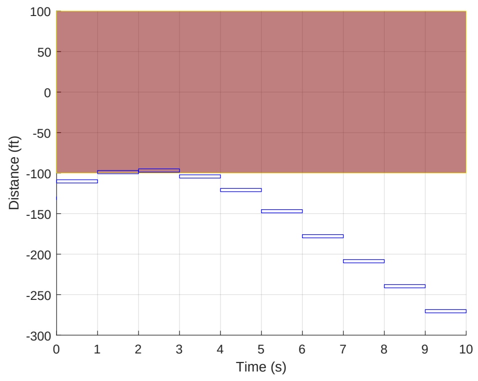
For the single pendulum it is sufficient to start with a smaller initial state to verify that the safety property is violated. NNV is able to compute the reach sets in 1.5 s. The reach sets along with random simulations are depicted in Figure 19.



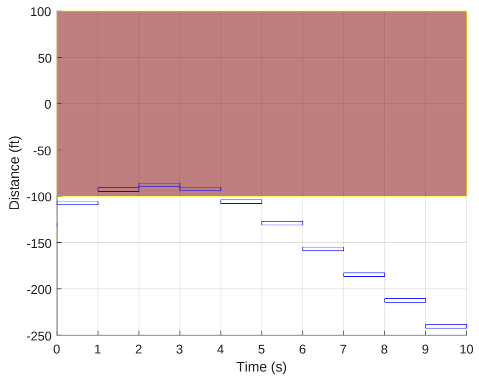
(a) 19.5



(b) 22.5



(c) 25.5



(d) 28.5

Figure 17: ***NNV.*** Analysis results for the VCAS benchmark showing the aircraft sets in blue and the unsafe region in red, when selecting the middle acceleration value at each control period

4.2.5 Double Pendulum

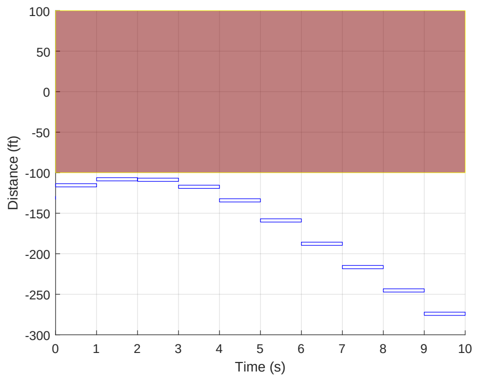
Similar to the single pendulum, we can verify that the property is violated starting from a smaller initial set. The results are depicted in Figure 20, and the computation times are 23.4 s for the less robust controller, and 23.2 s for the more robust controller.

4.2.6 Airplane

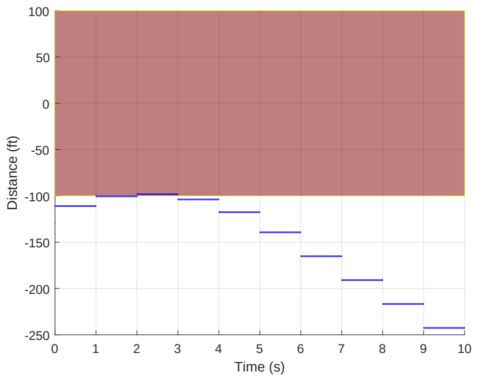
NNV is able to verify in 7 s that the property is violated by computing the reach sets from a smaller initial region. The results are depicted in Figure 21.

4.3 CORA

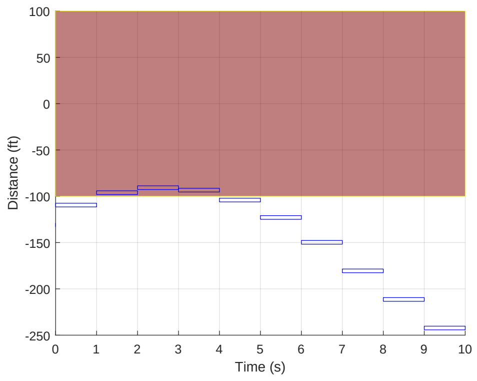
We present the results utilizing *CORA* on each of the benchmarks. CORA is able to show verification/violation of the specifications in all benchmarks except one using the entire input set without splitting. For details about the reachability parameters used, such as the step size of our algorithm for continuous time or the parameters for the propagation through the



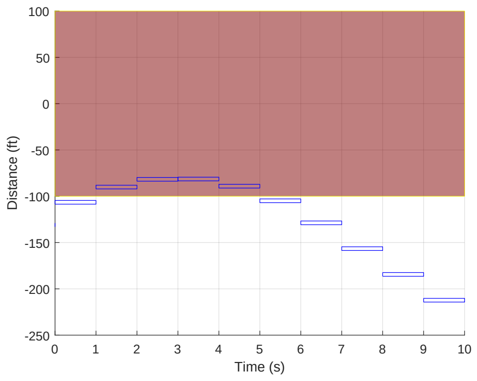
(a) 19.5



(b) 22.5



(c) 25.5



(d) 28.5

Figure 18: ***NNV.*** Analysis results for the VCAS benchmark showing the aircraft sets in blue and the unsafe region in red, when selecting the worst possible acceleration value at each control period.

neural network, we refer to the submission code available at `https://gitlab.com/goranf/ARCH-COMP/-/tree/master/2022/AINNCS/cora`. We provide exact times for each step in the verification process for better comparability. All time values are in seconds. In addition, we provide accompanying plots for each benchmark.

4.3.1 ACC

CORA is able to verify all specifications. The computed reachable set along with some simulations are shown in Figure 22.

```
Time to compute random simulations:  0.94224
Time to check violation by simulations:  0.0030243
Time to compute reachable set:  1.5869
Time to check verification:  0.049175
Result:  VERIFIED
Total Time:  2.5813
```

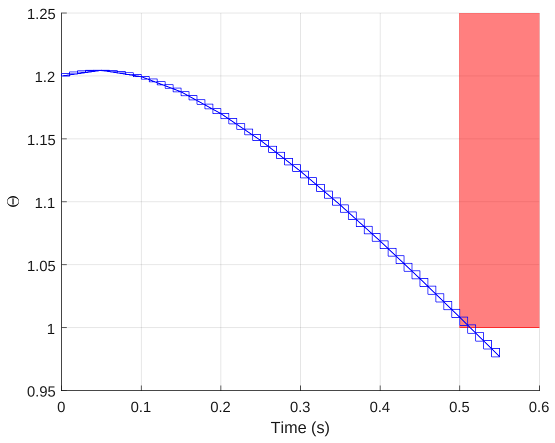
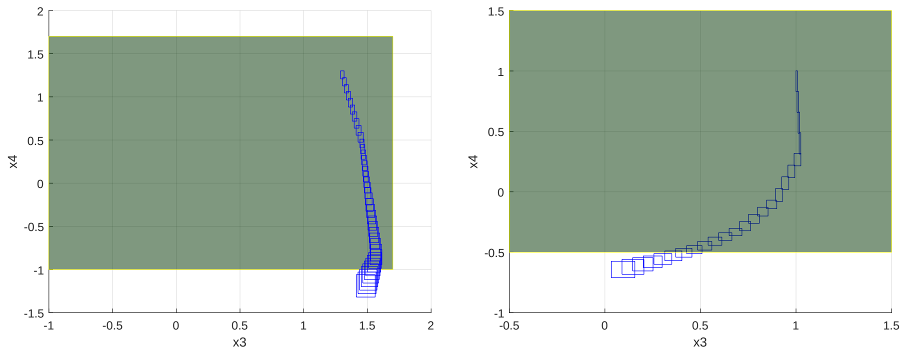


Figure 19: ***NNV.*** Analysis results for the single pendulum benchmark showing the sets in **blue** and the unsafe region in **red**.



(a) Less (b) More

Figure 20: ***NNV.*** Analysis results for the double pendulum benchmark with the *less* and *more* robust controllers, showing the reach sets in **blue** and the goal region in **green**.

4.3.2 Sherlock-Benchmark-9 (TORA)

The computed reachable set along with some simulations are shown in Figure 23.

Specification 1 CORA is able to verify the specifications from the original benchmark.

```
Time to compute random simulations:  0.43501
Time to check violation by simulations:  0.0089707
Time to compute reachable set:  10.6547
Time to check verification:  0.27649
Result:  VERIFIED
Total Time:  11.3752
```

Specification 2 CORA is able to verify specification 2 for both controllers.
Sigmoid controller:

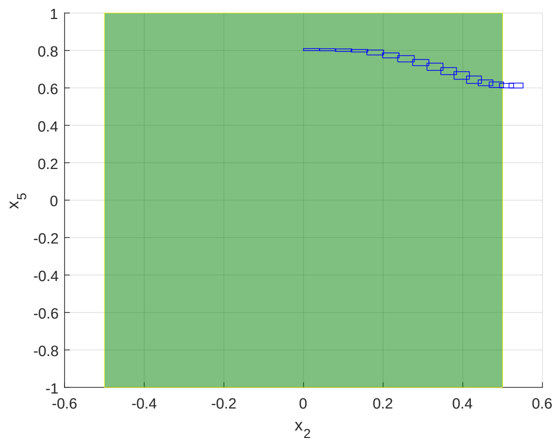


Figure 21: ***NNV.*** Analysis results for the airplane benchmark showing the reach sets in **blue** and the goal region in **green**.

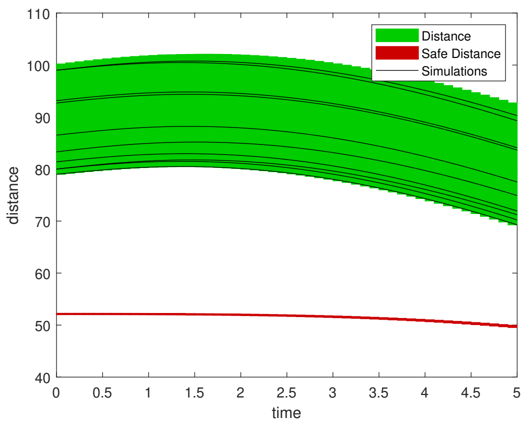


Figure 22: **CORA.** Analysis results for the ACC benchmark, including simulations.

```
Time to compute random simulations:  0.28046
Time to check violation by simulations:  0.0042218
Time to compute reachable set:  11.0289
Time to check verification:  0.056999
Result:  VERIFIED
Total Time:  11.3705
```

ReLU/Tanh controller:

```
Time to compute random simulations:  0.25347
Time to check violation by simulations:  0.003146
Time to compute reachable set:  12.126
Time to check verification:  0.06485
Result:  VERIFIED
Total Time:  12.4474
```

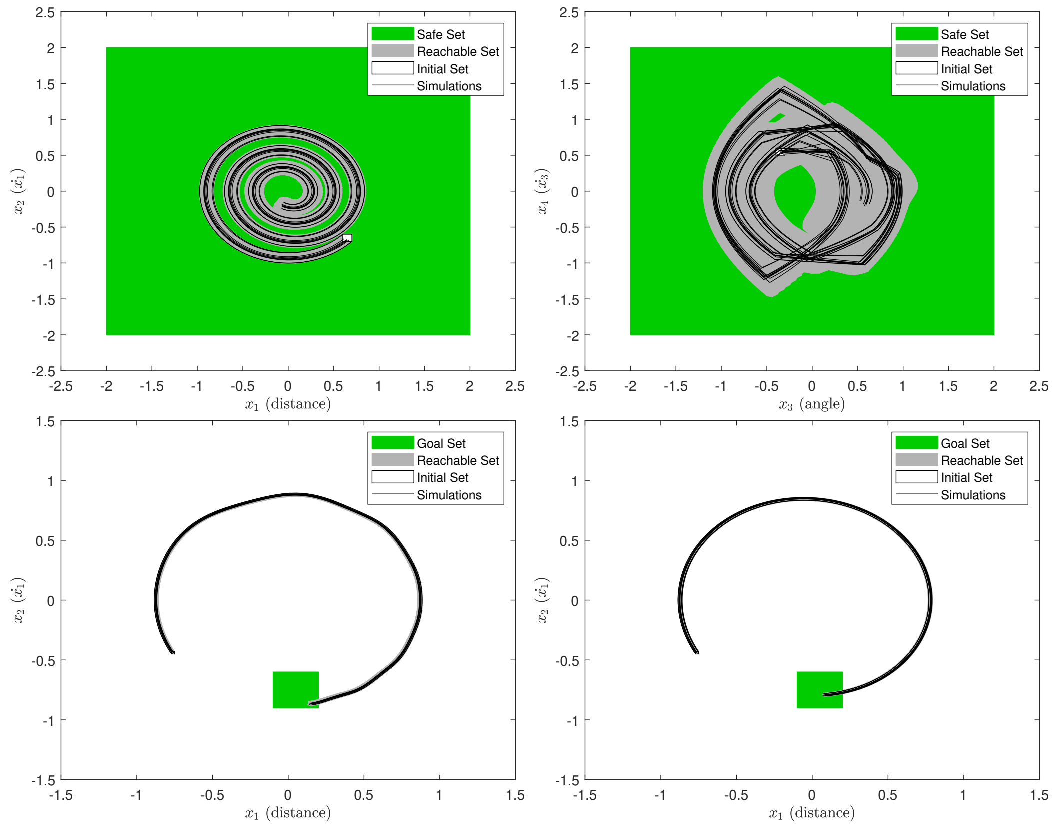


Figure 23: **CORA.** Analysis results for the Sherlock-Benchmark-9 (TORA) benchmark: specification 1 (top) and specifications 2 for the sigmoid (bottom left) and ReLU/tanh (bottom right) controller, including simulations.

4.3.3 Sherlock-Benchmark-10 (Unicycle)

CORA is able to verify the specifications of the Sherlock-Benchmark-10 (Unicycle) benchmark. The computed reachable set along with some simulations are shown in Figure 24.

```
Time to compute random simulations:  1.0836
Time to check violation by simulations:  0.012185
Time to compute reachable set:  2.6859
Time to check verification:  0.16118
Result:  VERIFIED
Total Time:  3.9429
```

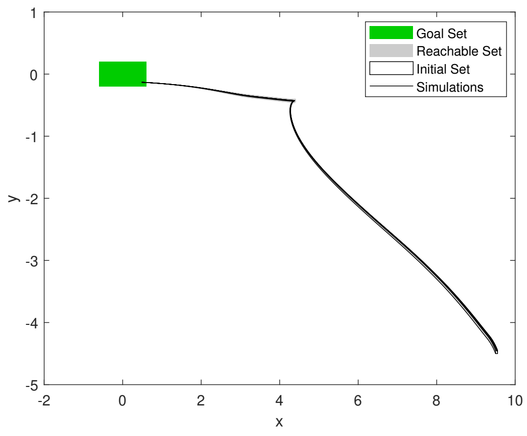


Figure 24: **CORA.** Analysis results for the Sherlock-Benchmark-10 (Unicycle) benchmark, including simulations.

4.3.4 VCAS

The VCAS benchmark has discrete time steps and multiple controllers, which is currently not supported by CORA. Thus, a custom algorithm was built for this benchmark. To deal with the discrete input set $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, we ran a simulation with each element of the input set individually. As proposed in the benchmark specifications, we show the results when always the middle acceleration of the controllers is chosen and the results when always the worst acceleration is chosen.

VCAS (middle acceleration) Here we always use the middle of the possible accelerations. We are able to verify the benchmark for $\dot{h}_0(0) \in \{-19.5, -22.5\}$ and can show violations for $\dot{h}_0(0) \in \{-25.5, -28.5\}$. The computed reachable set along with some simulations are shown in Figure 25. The times below are from the run with $\dot{h}_0(0) = -19.5$.

```
Time to compute random simulations:  0.015016
Time to check violation by simulations:  0.0029496
Time to compute reachable set:  0.03363
Time to check verification:  0.0011118
Result:  VERIFIED
Total Time:  0.052707
```

VCAS (worst acceleration) Here we always use the worst possible acceleration. We are able to verify the benchmark for $\dot{h}_0(0) \in \{-19.5\}$ and can show violations for $\dot{h}_0(0) \in \{-22.5, -25.5, -28.5\}$. The computed reachable set along with some simulations are shown in Figure 26. The times below are from the run with $\dot{h}_0(0) = -19.5$.

```
Time to compute random simulations:  0.013097
Time to check violation by simulations:  0.0011991
Time to compute reachable set:  0.026289
Time to check verification:  0.0022797
Result:  VERIFIED
Total Time:  0.042864
```

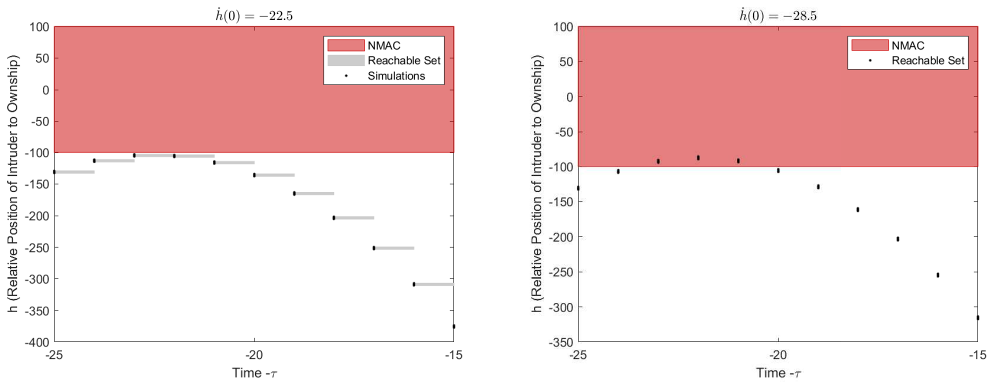


Figure 25: **CORA.** Simulation runs for the VCAS (middle acceleration) benchmark.

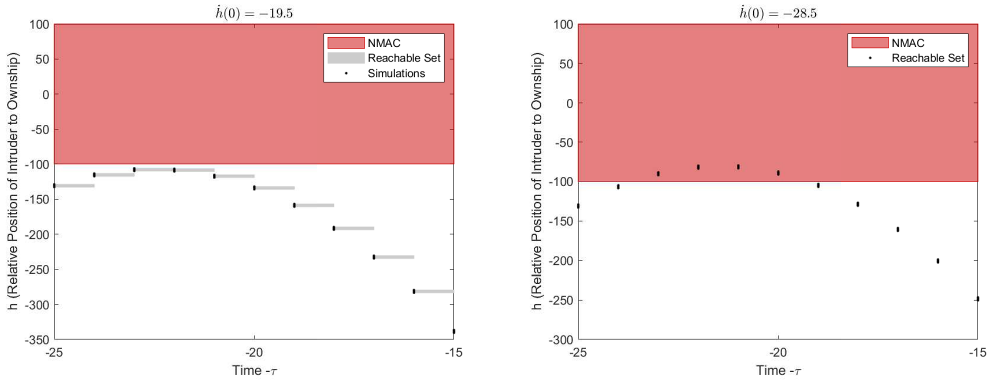


Figure 26: **CORA.** Simulation runs for the VCAS (worst acceleration) benchmark.

4.3.5 Single Pendulum

CORA is able to show specification violations of the single-pendulum benchmark. The simulation runs for this benchmark are shown in Figure 27.

```
Time to compute random simulations:  1.0298
Time to check violation by simulations:  0.0030485
Result:  VIOLATED
Total Time:  1.0328
```

4.3.6 Double Pendulum

We report the results from the double-pendulum benchmark.

Double Pendulum (less robust) CORA is able to show specification violations. The simulation runs for this benchmark are shown in Figure 28.

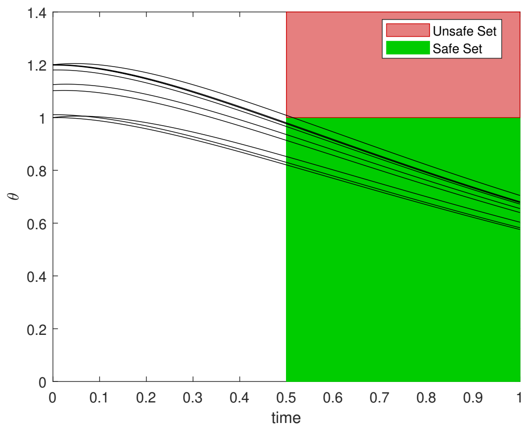


Figure 27: **CORA.** Simulation runs for the single pendulum benchmark.

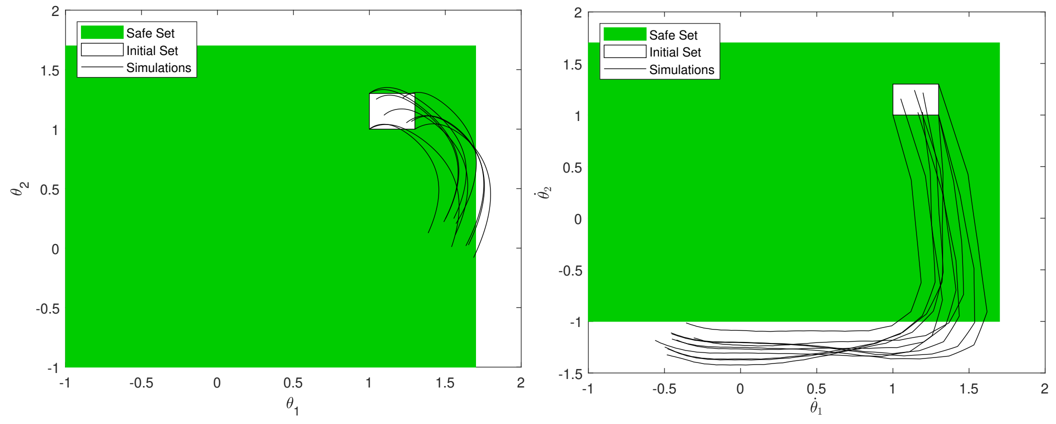


Figure 28: **CORA.** Simulation runs for the double pendulum (less robust) benchmark.

```
Time to compute random simulations:  0.7995
Time to check violation by simulations:  0.002714
Result:  VIOLATED
Total Time:  0.80222
```

Double Pendulum (more robust) CORA is able to show specification violations. The simulation runs for this benchmark are shown in Figure 29.

```
Time to compute random simulations:  0.57492
Time to check violation by simulations:  0.0023562
Result:  VIOLATED
Total Time:  0.57727
```

4.3.7 Airplane

CORA is able to show specification violations of the airplane benchmark. The simulation runs for this benchmark are shown in Figure 30.

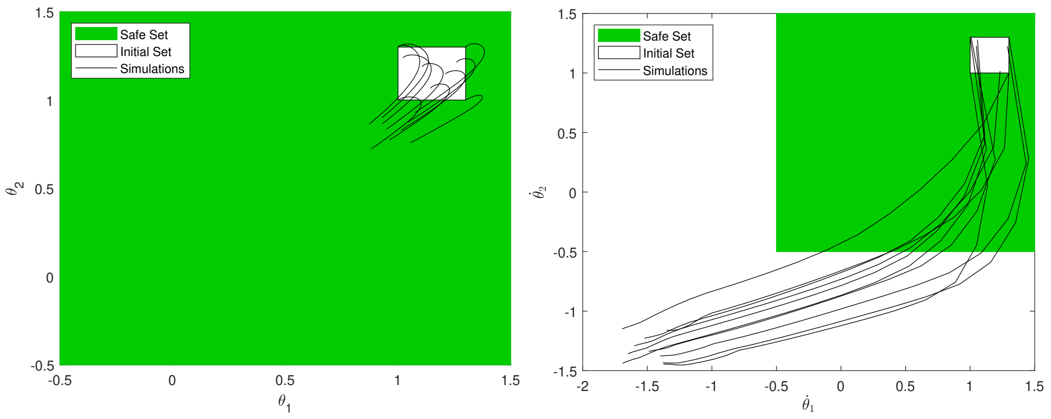


Figure 29: **CORA.** Simulation runs for the double pendulum (more robust) benchmark.

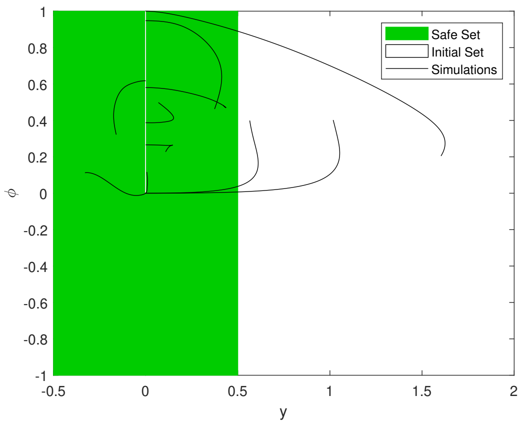


Figure 30: **CORA.** Simulation runs for the airplane benchmark.

```
Time to compute random simulations:  1.457
Time to check violation by simulations:  0.0032885
Result:  VIOLATED
Total Time:  1.4603
```

4.3.8 Attitude Control

CORA is able to verify the specifications of the attitude control benchmark. The computed reachable set along with some simulations are shown in Figure 31.

```
Time to compute random simulations:  0.55773
Time to check violation by simulations:  0.0053589
Time to compute reachable set:  1.0156
Time to check verification:  0.011554
Result:  VERIFIED
Total Time:  1.5902
```

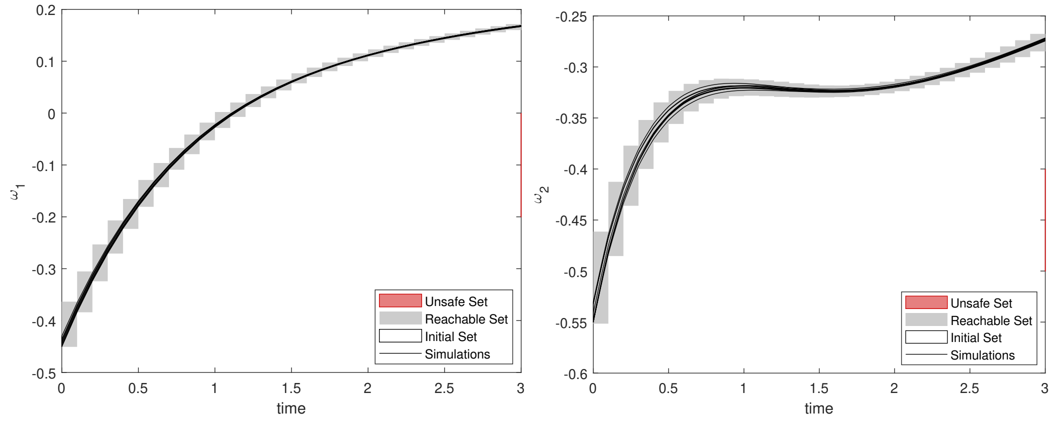


Figure 31: **CORA.** Analysis results for the attitude control benchmark, including simulations.

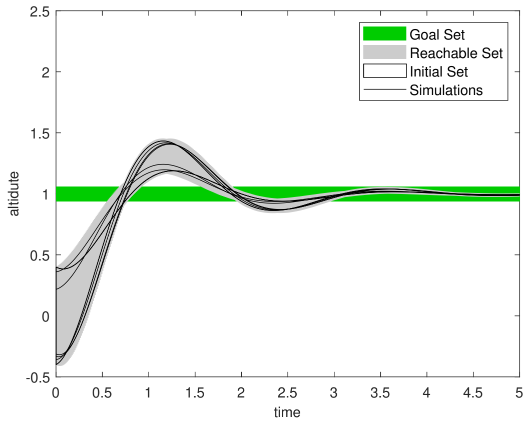


Figure 32: **CORA.** Analysis results for the QUAD benchmark, including simulations.

4.3.9 QUAD

CORA is able to verify the specifications of the QUAD benchmark by computing a tighter over-approximation of the neural network's output: The first nonlinear layer is approximated using a quadratic polynomial and an approximation error is added to obtain a sound over-approximation. All other nonlinear layers are approximated using linear polynomials. The computed reachable set along with some simulations are shown in Figure 32.

```
Time to compute random simulations:  0.86463
Time to check violation by simulations:  0.003554
Time to compute reachable set:  3941.286
Time to check verification:  0.34886
Result:  VERIFIED
Total Time:  3942.503
```

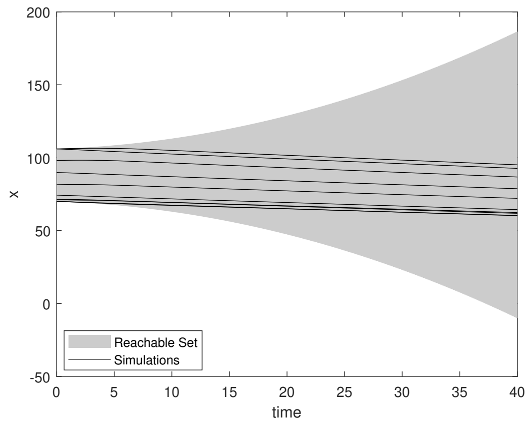


Figure 33: **CORA.** Analysis results for the spacecraft docking benchmark, including simulations.

4.3.10 Spacecraft Docking

The spacecraft docking benchmark appeared difficult to verify for our approach. Simulation runs seem to be stable, though. The computed reachable set along with some simulations are shown in Figure 33.

```
Time to compute random simulations:  1.5695
Time to check violation by simulations:  0.0084425
Time to compute reachable set:  4.1904
Time to check verification:  5.2892
Result:  UNKOWN
Total Time:  11.0576
```

4.4 POLAR

We present the experimental results utilizing **POLAR** on each of the benchmarks in Table 1. All of the results are *numerically conservative*. We skipped the VCAS benchmark, since some of the dynamics cannot be modeled by the current API of POLAR. The more detailed settings are given as follows.

ACC. This benchmark was verified by POLAR to be **safe** in 1.9 s, and the octagon overapproximation of the Taylor model (TM) flowpipes are presented in Figure 34.

Sherlock-Benchmark-9: TORA. There are two specifications given in this benchmark, and two neural network controllers in the second specification. For the first specification, the initial set is uniformly subdivided to $3 \times 3 \times 2 \times 2$ boxes, and the safety of the system is verified in 502 s. The reachable set overapproximations are shown in Figure 35. For the second specification, the verification tasks for the ReLU-tanh and the sigmoid controller take 0.14 s and 0.39 s, respectively, and the reachable sets are shown in Figure 36. No subdivision is performed.

Sherlock-Benchmark-10: Unicycle Car. POLAR verified the safety of this benchmark in 4.8 s without subdividing the initial set, and the result is given in Figure 37.

Single Pendulum. This benchmark was verified by POLAR to be **unsafe** in 0.1 s from the initial state $[1.2, 0.2]$, and the reachable set can be found in Figure 38.

Table 1: For each benchmark, we show the activation used in the neural network controller as σ, and the numbers of neuron in all the layers as **size**. In the POLAR setting, δ denotes the step-size, k_T denotes the order of Taylor models, k_B denotes the order of Bernstein approximations, and ε denotes the cutoff threshold, i.e., the threshold for truncating the small polynomial terms to the remainders. The runtime in seconds is shown in $T(s)$ column. 'Y': POLAR verifies the system is safe. 'U': the property is not verified. 'N': POLAR verifies the system is unsafe. *For docking, we do the verification for four different initial sets, and we demonstrate the average computation time here.

Benchmark	NN Controller		POLAR				
	σ	size	δ	k_T	k_B	ε	T (s)
ACC	ReLU	5x20x20x20x20x20	0.1	3	2	1e-6	**1.9 (Y)**
TORA	ReLU	100x100x100	0.025	7	2	1e-6	**502 (Y)**
	ReLU-tanh	20x20x20	0.1	4	2	1e-6	**0.14 (Y)**
	sigmoid	20x20x20	0.1	4	2	1e-6	**0.39 (Y)**
Unicycle Car	ReLU	500	0.05	3	2	1e-6	**4.8 (Y)**
Single Pendulum	ReLU	25x25	0.02	4	2	1e-6	**0.10 (N)**
Double Pendulum	ReLU (less)	25x25	0.01	4	2	1e-6	**0.51 (N)**
	ReLU (more)	25x25	0.01	4	2	1e-6	**0.24 (N)**
Airplane	ReLU	100x100x20	0.05	4	2	1e-7	**0.10 (N)**
Attitude Control	sigmoid	64x64x64	0.1	3	2	1e-5	**10.5 (Y)**
QUAD	sigmoid	64x64x64	0.005	4	4	1e-6	**647 (Y)**
Docking	tanh	4x256x256x4	0.05	4	2	1e-8	**57.4 (U)***

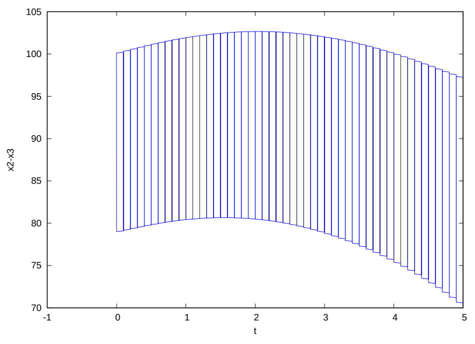


Figure 34: POLAR: ACC.

Double Pendulum. The reachability of this benchmark with a less robust and a more robust controller was verified by POLAR to be **unsafe** in 0.51 s and 0.24 s, respectively, when the initial state is $[1.3, 1.3, 1.3, 1.3]$. We show the computed reachable set in Figure 39.

Airplane. Starting from the initial state $[0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0]$, POLAR immediately proved the unsafety of the system.

Attitude Control. This benchmark was verified by POLAR to be **safe** in 10.5 s, and the reachable set can be found in Figure 40.

QUAD. This benchmark was verified by POLAR to be **safe** in 647 s, and the reachable sets are given in Figure 41.

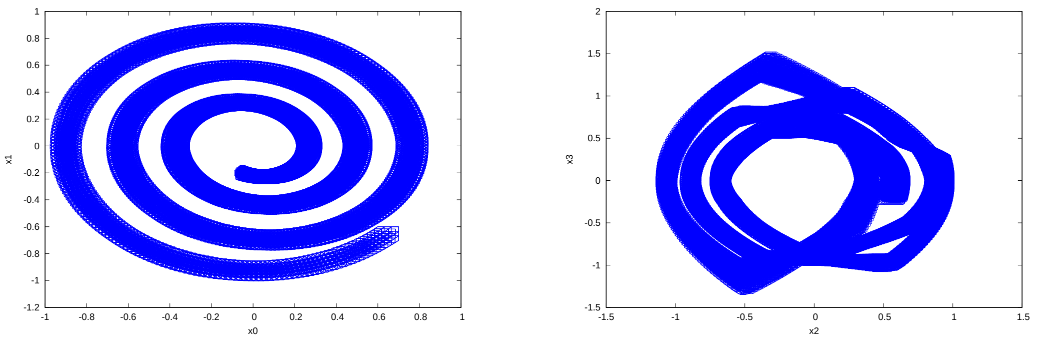


Figure 35: POLAR: Sherlock-B9: RELU network.

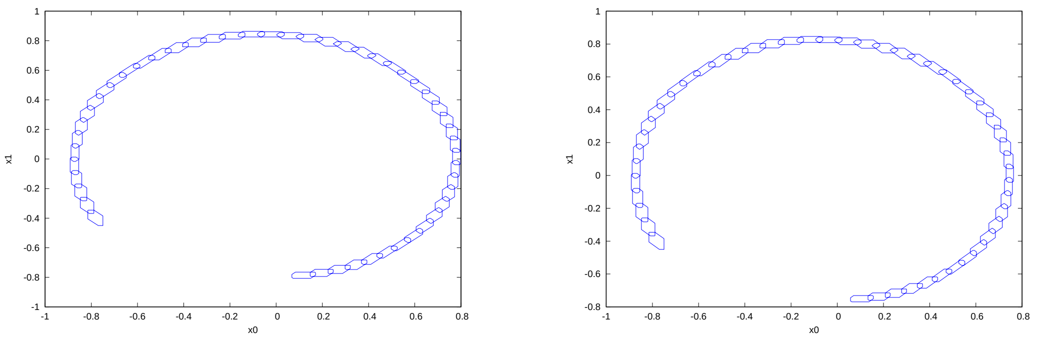
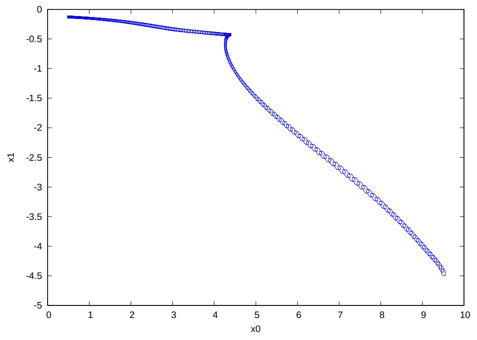


Figure 36: POLAR: TORA Heterogeneous.

Docking. The benchmark has not been fully verified. We consider four different corner initial sets and computed the reachable sets shown in Figure 42. Each of them costs around 57.4 s. No safety violation was found.

5 Category Status and Challenges

In the fourth iteration of the AINNCS category at ARCH-COMP, the participating tools CORA, JuliaReach, NNV and POLAR successfully analyzed different aspects of the benchmark problems, with the exception of the newly introduced *docking spacecraft*. This year's competition saw an improvement in the verified instances, going from 60% (2021) success rate to 75% (2022) success rate, despite the increasing benchmark complexity. In spite of some success analyzing the benchmarks in the AINNCS category, there are challenges remaining for the category, which we discuss these next along with some of the improvements with respect to previous competitions.



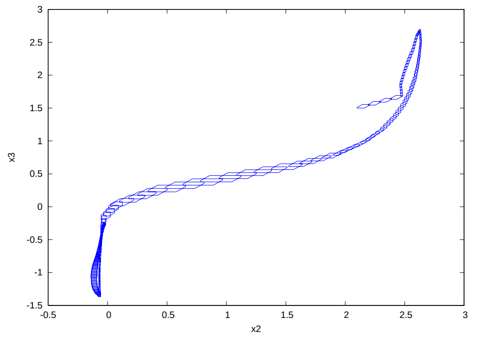


Figure 37: POLAR: Sherlock-Benchmark-10: Unicycle Car.

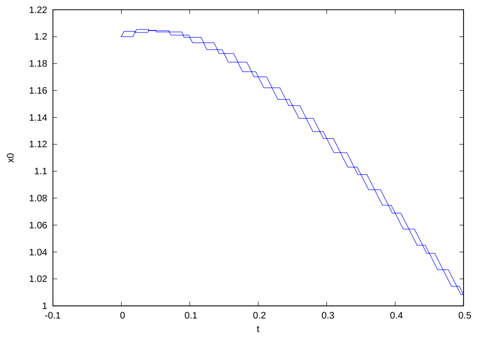


Figure 38: POLAR: Single Pendulum.

Hybrid Controllers: Some controllers involve a hybrid nature, such as the the VCAS benchmark. This is a very complex control system formed by 9 different neural networks that are chosen based on plant's states. These controllers have also a bang-bang output characteristic, meaning that the output range is not continuous, but is chosen from a discrete set of values depending on the current neural network executed, as well as all output values and the aircraft states. We observe that 3 out of 4 tools successfully verified this benchmark, which is an improvement from previous iterations, i.e. 2 out of 3 of the tools were able to verify the system last year compare to 3 out of 4 this year.

Activation Function Types (controllers): For this year's set of benchmarks, all neural network controllers contain one or more of the following activation functions: ReLU, linear, sigmoid, and tanh, same as last year. The main difference is the improvement in support for each activation function, as all 4 tools have support for all the types, linear, piecewise linear (ReLU) and nonlinear activation functions.

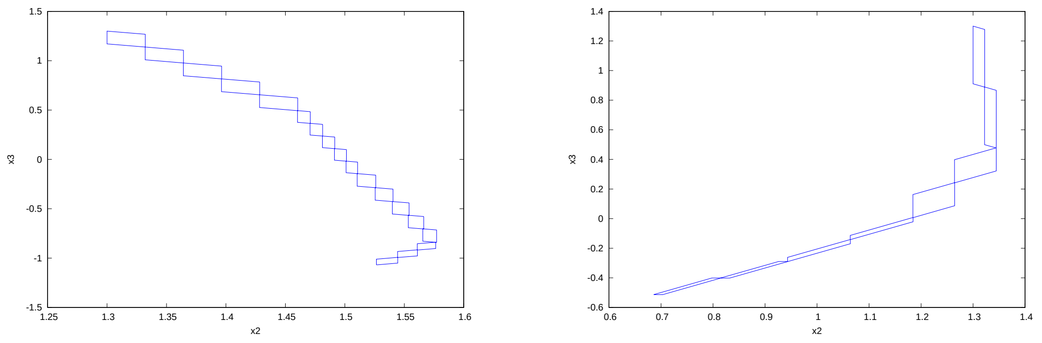


Figure 39: POLAR: Double Pendulum.

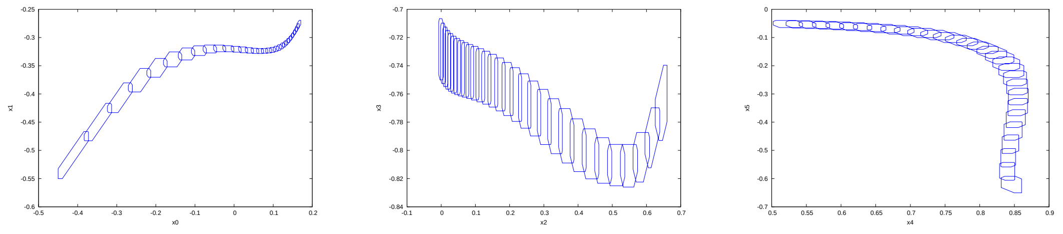


Figure 40: POLAR: Attitude Control.

Plant Models: This year we have considered linear and nonlinear plants, both in discrete and continuous time. We plan to add hybrid automata plants in future iterations, as we look to report a more complete analysis of the participating verification tools. Hybrid automata plants will be especially interesting with the complex nature of combined continuous and discrete dynamics, which is very challenging for current AINNCS verification tools.

Neural Network Architectures and Parameterization: Similar to last year, when we compare the neural network architectures presented in this work with some of the networks that can be analyzed in absence of the plant, these are fairly simple, in the sense none of the networks have more than a thousand neurons, and none exceed 5 hidden layers in their architecture. Also, the maximum number of inputs and outputs of the controllers are 12 and 6, respectively, in the airplane benchmark. If we consider the VCAS benchmark, these networks have 9 outputs, although these are translated into a single input to the plant model. However, for some benchmarks, there are still state-space explosion and scalability issues to address in both the neural network controllers and plant analysis.

Time horizons: Similar to previous iterations, all the tools performed bounded (finite) time horizon verification analysis, also known as bounded model checking, where the main difference is that all the participating tools rely on reachability analysis methods to analyze safety. Alternative approaches for performing unbounded (infinite) time horizon verification

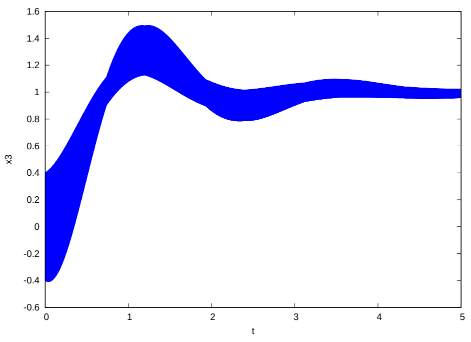
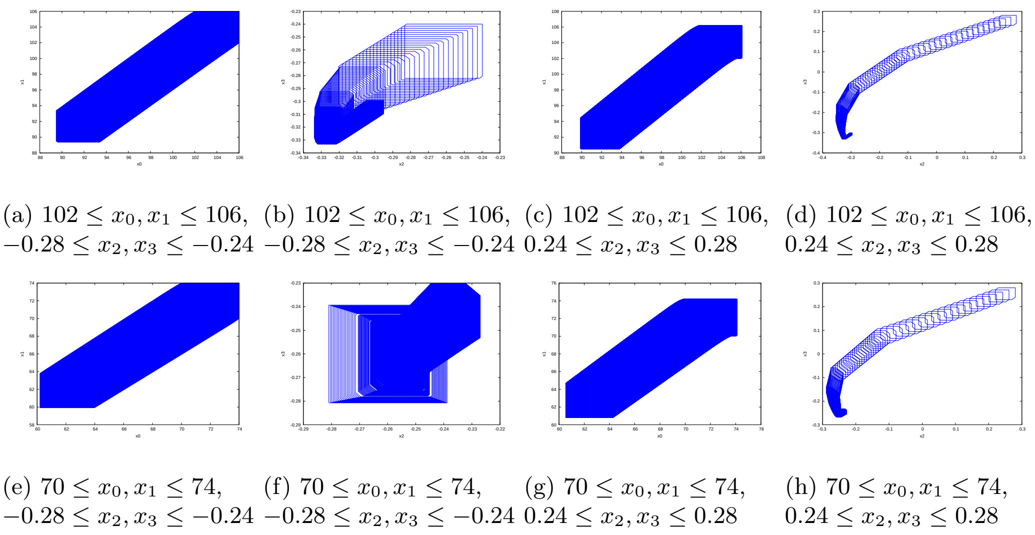


Figure 41: POLAR: QUAD.



(a) $102 \leq x_0, x_1 \leq 106$, $-0.28 \leq x_2, x_3 \leq -0.24$ (b) $102 \leq x_0, x_1 \leq 106$, $-0.28 \leq x_2, x_3 \leq -0.24$ (c) $102 \leq x_0, x_1 \leq 106$, $0.24 \leq x_2, x_3 \leq 0.28$ (d) $102 \leq x_0, x_1 \leq 106$, $0.24 \leq x_2, x_3 \leq 0.28$

(e) $70 \leq x_0, x_1 \leq 74$, $-0.28 \leq x_2, x_3 \leq -0.24$ (f) $70 \leq x_0, x_1 \leq 74$, $-0.28 \leq x_2, x_3 \leq -0.24$ (g) $70 \leq x_0, x_1 \leq 74$, $0.24 \leq x_2, x_3 \leq 0.28$ (h) $70 \leq x_0, x_1 \leq 74$, $0.24 \leq x_2, x_3 \leq 0.28$

Figure 42: POLAR: Docking.

exist, such as those building on barrier certificates, a form of continuous analog of the classical inductive invariance proof rule. The existing methods could incorporate invariance checks on the computed reachable states to attempt to determine if the reachability analysis reaches a fixed-point (if the reachability analysis terminates, which for the class of systems considered, is not guaranteed as the reachability analysis with nonlinear plants is undecidable). However, no current methods evaluated in the competition utilize this approach, and this is a promising avenue for future work to provide guarantees beyond finite time horizons.

Model Formats: Similar to last year, we have found more useful and convenient to simply share the plant models in a plain format, such as MATLAB functions, where the participants could easily extract the ODEs. As for the neural network models, we provide them in the ONNX format[3], .mat format[4], and the original format used by proposer of the benchmark. ONNX format was very convenient, as most of the participating tools have integrated ONNX into their frameworks this year. However, we found that there are still discrepancies among the different versions and frameworks these ONNX models were created from (e.g., different input/output transformations are not always supported by every framework as experienced on the *Docking spacecraft* benchmark). Having a unified ONNX version remains a challenge, but the community is closer to achieving this goal with the continuous development of these ONNX importers. In addition, initiatives more focused on neural network verification, such as VNN-LIB[5] and VNN-COMP[6], may help toward this goal. In terms of specifications, we are planning to make use of the vnnlib format to define the formal specifications for each benchmark in a similar manner that VNN-COMP has used for this year's competition to further automate the verification process.

6 Conclusion and Outlook

This report presents the results on the fourth ARCH friendly competition for closed-loop systems with neural network controllers. For this edition, four tools have participated and attempted to solve 10 benchmarks: CORA, JuliaReach, NNV, and POLAR. The problems elucidated in this paper are challenging and diverse; the presented results probably provide the most complete assessment of current tools for the safety verification in AINNCS. The report provides a good overview of the intellectual progression of this rapidly growing field, and it is our hope to stimulate the development of efficient and effective methods capable of use in real-world applications. In the past three years, the complexity of the benchmarks has consistently increased along with the capabilities of the participant tools, leading to the most challenging competition and the best verification results thus far, which is a good indicator for this growing and maturing field. This has been achieved by to the continuous development and improvements in some of the tools that have participated in previous iterations (NNV, JuliaReach), and new tools being developed every year, such as the recent CORA and POLAR that help push this field forward.

7 Acknowledgments

The material presented in this report is based upon work supported by the National Science Foundation (NSF) under grant numbers FMiTF 2220401, 2220426, 2220426, EPCN 2028001, CCF 1646497, CCF 1834324, CNS 1834701, IIS 1724341, the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089, the US Air Force Research Laboratory (AFRL) under contract number FA8650-16-C-2642, the Air Force Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019, and the Office of Naval Research (ONR) grant N00014-19-1-2496. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any

[3]Open Neural Network Exchange: `https://github.com/onnx/onnx`
[4]Direct input format used by *NNV* without transformation.
[5]`http://www.vnnlib.org/`
[6]`https://github.com/verivital/vnn-comp/`

copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, AFRL, DARPA, or NSF. Luis Benet acknowledges support from PAPIIT grant IG-101122. Christian Schilling acknowledges the support from DIREC - Digital Research Centre Denmark and the Villum Investigator Grant S4OS. Tobias Ladner gratefully acknowledges financial support from the project FAI funded by the German Research Foundation (DFG) under project number 286525601.

A Specification of Used Machines

A.1 $\mathbf{M}_{JuliaReach}$

- Processor: Intel Core i7-10610U CPU @ 1.80GHz x64
- Memory: 32 GB
- OS: Ubuntu 22.04

A.2 $\mathbf{M}_{CORA}$

- Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
- Memory: 64 GB
- OS: Windows 11

A.3 $\mathbf{M}_{nnv}$

- Processor: AMD Ryzen 9 5900X @ 3.70 GHz x 12 (x64)
- Memory: 64 GB
- OS: Windows 10

A.4 $\mathbf{M}_{POLAR}$

- Processor: Apple M1 @ 3.2 GHz
- Memory: 8 GB
- OS: macOS Monterey

References

[1] *Proceedings of the 7th International Conference On Formal Methods In Software Engineering, FormaliSE 2019, collocated with ICSE 2019, Montréal, Canada, May 27, 2019*. ACM, 2019.

[2] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[3] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[4] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of Taylor models in CORA 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 2018.

[5] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 719–726, 2017.

[6] Rajeev Alur. Formal Verification of Hybrid Systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 273–278, New York, NY, USA, 2011. ACM.

[7] S. Bak, S. Bogomolov, T. A. Henzinger, T. T. Johnson, and P. Prakash. Scalable static hybridization methods for analysis of nonlinear systems. In *Proc. of the 19th ACM International Conference on Hybrid Systems: Computation and Control*, pages 155–164, 2016.

[8] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. Hyst: A source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 128–133, New York, NY, USA, 2015. ACM.

[9] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. 2021.

[10] Randal W. Beard. Quadrotor dynamics and control. Technical report, 2008.

[11] Luis Benet, Marcelo Forets, David P. Sanders, and Christian Schilling. Taylormodels.jl: Taylor models in julia and its application to validated solutions of ODEs. In *SWIM*, 2019.

[12] Luis Benet and David P. Sanders. TaylorSeries.jl: Taylor expansions in one and several variables in Julia. *Journal of Open Source Software*, 4(36):1043, 2019.

[13] Luis Benet and David P. Sanders. JuliaDiff/TaylorSeries.jl. `https://github.com/JuliaDiff/TaylorSeries.jl`, 2021.

[14] Luis Benet and David P. Sanders. JuliaIntervals/TaylorModels.jl. `https://github.com/JuliaIntervals/TaylorModels.jl`, 2021.

[15] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. JuliaReach: a toolbox for set-based reachability. In *HSCC*, pages 39–44. ACM, 2019.

[16] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013.

[17] Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS'16)*, pages 13–24. IEEE Computer Society, 2016.

[18] W. H. CLOHESSY and R. S. WILTSHIRE. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences*, 27(9):653–658, 1960.

[19] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019.*, pages 157–168, 2019.

[20] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151 – 156, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[21] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A Dual Approach to Scalable Verification of Deep Networks. *CoRR*, abs/1803.06567, 2018.

[22] Jiameng Fan, Chao Huang, Wenchao Li, Xin Chen, and Qi Zhu. Towards verification-aware knowledge distillation for neural-network controlled systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[23] Jiameng Fan, Chao Huang, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN*: A tool for reachability analysis ofneural-network controlled systems. In *to appear on International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2020.

[24] Marcelo Forets and Christian Schilling. LazySets.jl: Scalable symbolic-numeric set computations. *Proceedings of the JuliaCon Conferences*, 1(1):11, 2021.

[25] G. W. Hill. Researches in the lunar theory. *American Journal of Mathematics*, 1(1):5–26, 1878.

[26] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In *To appear on International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022.

[27] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[28] M. Jankovic, D. Fontaine, and P. V. Kokotovic. Tora example: cascade- and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3):292–297, May 1996.

[29] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[30] Taylor T. Johnson, Diego Manzanas Lopez, Luis Benet, Marcelo Forets, Sebasti\'an Guadalupe, Christian Schilling, Radoslav Ivanov, Taylor J. Carpenter, James Weimer, and Insup Lee. Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 90–119. EasyChair, 2021.

[31] Taylor T Johnson, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, and Chao Huang. Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 107–139. EasyChair, 2020.

[32] K. D. Julian and M. J. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *CoRR*, abs/1903.00520, 2019.

[33] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[34] Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Transactions on Automatic Control*, 66(9):4043–4058, 2020.

[35] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open-and closed-loop neural network verification using polynomial zonotopes. *arXiv preprint arXiv:2207.02715*, 2022.

[36] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[37] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A Survey of Deep Neural Network Architectures and their Applications. *Neurocomputing*, 234:11 – 26, 2017.

[38] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 103–119. EasyChair, 2019.

[39] Amir Maleki and Chelsea Sindrane. Benchmark examples for ainncs-2020, 2020.

[40] MathWorks. *Adaptive Cruise Control System* block. `https://www.mathworks.com/help/mpc/ref/adaptivecruisecontrolsystem.html`, 2018.

[41] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[42] Jorge A. Pérez-Hernández and Luis Benet. PerezHz/TaylorIntegration.jl. `https://github.com/PerezHz/TaylorIntegration.jl`, 2021.

[43] S. Prajna, P.A. Parrilo, and A. Rantzer. Nonlinear control synthesis by convex optimization. volume 49, pages 310–314, 2004.

[44] S. Joe Qin and Thomas A. Badgwell. An overview of nonlinear model predictive control applications. In Frank Allgöwer and Alex Zheng, editors, *Nonlinear Model Predictive Control*, pages 369–392, Basel, 2000. Birkhäuser Basel.

[45] Umberto J. Ravaioli, James Cunningham, John McCarroll, Vardaan Gangal, Kyle Dunlap, and Kerianne L. Hobbs. Safe reinforcement learning benchmark environments for aerospace control systems. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–20, 2022.

[46] Stuart Russell, Daniel Dewey, and Max Tegmark. Research Priorities for Robust and Beneficial Artificial Intelligence. *AI Magazine*, 36(4):105, 2015.

[47] Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. Verification of neural-network control systems by integrating Taylor models and zonotopes. In *AAAI*, pages 8169–8177. AAAI Press, 2022.

[48] Malcolm D. Shuster. Survey of attitude representations. *Journal of the Astronautical Sciences*, 41(4):439–517, October 1993.

[49] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In *NeurIPS*, pages 10825–10836, 2018.

[50] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Kevin Leyton-Brown, David C. Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller. "artificial intelligence and life in 2030." one hundred year study on artificial intelligence: Report of the 2015-2016 study panel, 2016.

[51] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[52] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2020.

[53] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T. Johnson, and Xenofon Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Trans. Embed. Comput. Syst.*, 18(5s), October 2019.

[54] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 670–686. Springer International Publishing, 2019.

[55] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[56] Weiming Xiang and Taylor T. Johnson. Reachability analysis and safety verification for neural network control systems. *CoRR*, abs/1805.09944, 2018.

[57] Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel A. Rosenfeld, and Taylor T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *CoRR*, abs/1810.01989, 2018.

[58] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *CoRR*, abs/1708.03322, 2017.

[59] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *CoRR*, abs/1712.08163, 2017.

[60] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2018.

[61] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Specification-guided safety verification for feedforward neural networks. *CoRR*, abs/1812.06161, 2018.

[62] Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, and Taylor T. Johnson. Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–10, 2020.