EPiC Series in Computing
Volume 80, 2021, Pages 90–119
8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)

ARCH-COMP21 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants

Taylor T. Johnson[1], Diego Manzanas Lopez[1] Luis Benet[2], Marcelo Forets[3], Sebastián Guadalupe[3], Christian Schilling[4,5], Radoslav Ivanov[6], Taylor Carpenter[6], James Weimer[6], Insup Lee[6]

[1] Vanderbilt University
Nashville, TN
{taylor.johnson, diego.manzanas.lopez }@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, sebastianguadalupe00@gmail.com
[4] University of Konstanz
Konstanz, Germany
[5] Aalborg University
Aalborg, Denmark
christianms@cs.aau.dk
[6] University of Pennsylvania
Philadelphia, PA
{rivanov, carptj, weimerj, lee}@seas.upenn.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 2021. In the third edition of this AINNCS category at ARCH-COMP, three tools have been applied to solve seven different benchmark problems, (in alphabetical order): JuliaReach, NNV, and Verisig. JuliaReach is a new participant in this category, Verisig participated previously in 2019 and NNV has participated in all previous competitions. This report is a snapshot of the current landscape of tools and the types of benchmarks for which these tools are suited. Due to the diversity of problems, lack of a shared hardware platform, and the early stage of the competition, we are not ranking tools in terms of performance, yet the presented results combined with 2020 results probably provide the most complete assessment of current tools for safety verification of NNCS.

1 Introduction

Neural Networks (NNs) have demonstrated an impressive ability for solving complex problems in numerous application domains [44]. The success of these models in contexts such as adaptive control, non-linear system identification [31], image and pattern recognition, function approximation, and machine translation, has stimulated the creation of technologies that are directly impacting our everyday lives [37], and has led researchers to believe that these models possess the power to revolutionize a diverse set of arenas [35].

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 [27]. In a study by Szegedy et al. [38], 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 [44]. 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 [26], 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 ([29, 17, 5]). 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 [15, 40]. 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) [8], 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 2021 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 third iteration of the AINNCS category at ARCH-COMP and builds on the previous iterations and reports from 2019 and 2020 [30, 24]. Participating tools are summarized in Sec. 2. Please, see [44] 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.

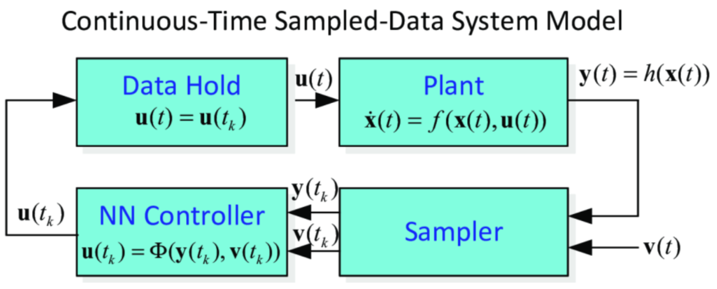
Continuous-Time Sampled-Data System Model
Data Hold
$\mathbf{u}(t) = \mathbf{u}(t_k)$
$\mathbf{u}(t)$
Plant
$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$
$\mathbf{y}(t) = h(\mathbf{x}(t))$
$\mathbf{u}(t_k)$
NN Controller
$\mathbf{u}(t_k) = \Phi(\mathbf{y}(t_k), \mathbf{v}(t_k))$
$\mathbf{y}(t_k)$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

Since 2019, we have seen significant progress on the verification algorithms, tools developed and benchmarks verified. Seven different tools have participated in the past three years while adding and improving the different reachability methods such as Verisig [20, 21] and NNV [42].

[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 JuliaReach, NNV, and Verisig. The tools participating in this AINNCS category *Artificial Intelligence / Neural Network Control Systems in Continuous and Hybrid Systems Plants* are introduced subsequently in alphabetical order.

JuliaReach (Luis Benet, Marcelo Forets, Sebastián Guadalupe, Christian Schilling) JuliaReach [13] 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 brand-new package for reachability analysis of neural-network controlled systems is NeuralNetworkAnalysis.jl. This package handles the closed-loop analysis and queries sub-problems to other libraries, namely to ReachabilityAnalysis.jl for continuous-time analysis of the plant models and NeuralVerification.jl [28] for the neural-network analysis. Additional set computations are performed with LazySets.jl [18]. JuliaReach can also run parallel simulations, where numerical integration of differential equations is handled by DifferentialEquations.jl [34].

For the plant analysis we use the sound algorithm `TMJets` based on interval arithmetic and Taylor models, which is implemented in Taylor models [9, 12], which itself integrates TaylorSeries.jl [10, 11] and TaylorIntegration.jl [32]. 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 the neural-network analysis we use an abstract interpretation based on zonotopes [36]. Due to careful conversions, JuliaReach can preserve high precision even after many control periods. Still, sometimes JuliaReach partitions the initial states to increase precision. Splitting into independent subproblems enables parallelization, but in this category JuliaReach executes sequentially. For falsification, JuliaReach chooses an initial point but still uses set-based analysis since, although most benchmark models are deterministic, non-validated simulations may yield wrong results.

NNV NNV (Neural Network Verification Tool) [42, 39, 41, 47, 49, 46, 45, 43, 48, 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 [7, 6] 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`.

Verisig Verisig [21, 19, 22, 20] is a tool for verifying safety properties of closed-loop hybrid systems with deep neural network (DNN) components. Verisig supports sigmoid and tanh-based networks and exploits the fact that the sigmoid (and the tanh) is the solution to a quadratic differential equation, which allows us to transform the neural network into an equivalent hybrid system. By composing the network's hybrid system with the plant's, Verisig transforms the problem into a hybrid system verification problem which can be solved using state-of-the-art reachability tools. We have performed a number of extensions to improve the scalability and accuracy of approach, including Taylor model preconditioning and shrink wrapping, as well as a parallelizable implementation in C++. Verisig is implemented in conjunction with Flow* [14], in order to allow for a smooth transition between the NN and the hybrid system. Verisig is available at https://github.com/verisig.

3 Benchmarks

For the competition, we have selected seven benchmarks. A few of them, such as the TORA benchmark, are presented with several different controllers that can be analyzed. The selected benchmarks are the same as last year's competition. We now describe these benchmarks in no particular order and we have made them readily available online.[2]

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. The neural network computes optimal control actions while satisfying safe distance, velocity, and acceleration constraints using model predictive control (MPC) [33]. 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 as follows:

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

where x_i is the position, v_i is the velocity, γ_i is the acceleration of the car, a_i is the acceleration control input applied to the car, and $u = 0.0001$ is the friction control where $i \in \{\text{ego, lead}\}$. For this benchmark we have developed four neural network controllers with 3, 5, 7, and 10 hidden layers of 20 neurons each, although we only evaluate the one with 5 layers. All of them have the same number of inputs (v_{set}, T_{gap}, v_{ego}, D_{rel}, v_{rel}), and one output (a_{ego}).

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_{lead} = -2. We want to check whether there is a collision in the following 5 seconds. Formally, this safety specification of the system can be expressed as $D_{\text{rel}} = x_{\text{lead}}$ - $x_{\text{ego}} \geq D_{\text{safe}}$, where $D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \times v_{\text{ego}}$, and $T_{\text{gap}} = 1.4$ seconds and $D_{\text{default}} = 10$. The initial conditions are: $x_{\text{lead}}(0) \in [90,110]$, $v_{\text{lead}}(0) \in [32,32.2]$, $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in [30, 30.2]$, $x_{\text{ego}} \in [10,11]$. A control period of 0.1 seconds is used.

3.2 Sherlock-Benchmark-9 (TORA)

This benchmark is that of a TORA (translational oscillations by a rotational actuator) [15, 23]. The model is that of a cart 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 4 dimensional system, given by the following equations :

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

[2] https://github.com/verivital/ARCH-COMP2021

A neural network controller was trained for this system, using data-driven model predictive controller proposed in [16]. The trained network had 3 hidden layers, with 100 neurons in each layer making a total of 300 neurons. 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 was $1s$.

Specification The verification problem here is that of safety. 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 stay within the box $\mathbf{x} \in [-2, 2]^4$, for a time window of $20s$.

3.3 Sherlock-Benchmark-10 (Unicycle Car Model)

This benchmark is that of a unicycle model of a car [15]. It models the dynamics of a car involving 4 variables, specifically the x and y coordinates on a 2 dimensional plane, as well as velocity magnitude (speed) and steering angle.

$$\dot{x_1} = x_4 \cos(x_3), \dot{x_2} = x_4 \sin(x_3), \dot{x_3} = u_2, \dot{x_4} = u_1 + w, \tag{3}$$

where w is a bounded error in the range $[-1e-4, 1e-4]$. A neural network controller was trained for this system, using a model predictive controller as a "demonstrator" or "teacher". The trained network has 1 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 was $0.2s$.

Specification The verification problem here is that of reachability. 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 $x_4 \in [1.5, 1.51]$, it is required to prove that the system reaches 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 $10s$.

3.4 VCAS Benchmark

This benchmark is a closed-loop variant of *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 [25]. Once 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 seconds 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.

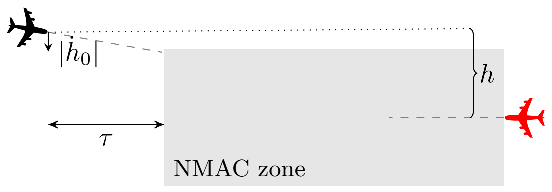

Figure 2: VerticalCAS encounter geometry

There are 9 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 adv 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 (1), 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: (2) DNC: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$; (3) DND: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$; (4) DES1500: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$; (5) CL1500: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$; (6) SDES1500: $\{-\frac{g}{3}\}$; (7) SCL1500: $\{\frac{g}{3}\}$; (8) SDES2500: $\{-\frac{g}{3}\}$; (9) 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 $(h(t+1), \dot{h}_0(t+1), \tau(t+1), \text{adv}(t+1))$ can be computed as follows:

$$\begin{aligned} h(t+1) &= h - \dot{h}_0 \Delta\tau - 0.5 \ddot{h}_0 \Delta\tau^2 \\ \dot{h}_0(t+1) &= \dot{h}_0 + \ddot{h}_0 \Delta\tau \\ \tau(t+1) &= \tau - \Delta\tau \\ \text{adv}(t+1) &= \text{adv}' \end{aligned}$$

where $\Delta\tau = 1$.

Specification For this benchmark the aim is to verify that the ownship is 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 as follows: $h(0) \in [-133, -129]$, $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, $\tau(0) = 25$ and $\text{adv}(0) = \text{COC}$.

3.5 Single Pendulum Benchmark

This is the classical inverted pendulum environment. 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 exists with a coefficient of c. The governing equation of motion can be obtained as:

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

where θ is the angle that link makes with the upward vertical axis, and $\dot{\theta}$ is the angular velocity. The state vector is:

$$[\theta, \dot{\theta}] \tag{5}$$

Controllers are trained using *behavior cloning*, a supervised learning approach for training controllers. Here, a neural network is trained to replicate expert demonstrations. We initially generate a set of *expert* control inputs for trajectories originating from different initial states of the system. *Expert* control inputs are defined as those that lead the system to reach to its goal state in finite time. The expert control inputs are generated using optimal control techniques. Specifically, we have used an implementation of LQR (Linear Quadratic Regulator) and iLQR (iterative LQR) control. The code for these implementations, as well as training procedures, are provided.

The continuous-time equations of motion may be written as a series of first order ODEs where $x_1 = \theta$ and $x_2 = \dot{\theta}$:

$$\dot{x}_1 = x_2 \tag{6a}$$

$$\dot{x}_2 = \frac{g}{L} \sin x_1 + \frac{1}{mL^2}\left(T - c\, x_2\right) \tag{6b}$$

The difference equations for the discrete time version of the system are obtained by using forward Euler integration:

$$x_{1_{t+1}} = x_{1_t} + \dot{x}_{1_t}\Delta t \tag{7a}$$

$$x_{2_{t+1}} = x_{2_t} + \dot{x}_{2_t}\Delta t \tag{7b}$$

The model involves several parameters, as follows.

$$m = 0.5, L = 0.5, c = 0., g = 1.0$$

The controller timestep (and dynamics timestep for discrete time) for `controller single pendulum` is $\Delta t = 0.05$. The initial set is

$$[\theta, \dot{\theta}] = [1.0, 1.2] \times [0.0, 0.2].$$

Specification The discrete-time safety specification is: $\forall n_t : 10 \leq n_t \leq 20, \theta \in [0.0, 1.0]$. The continuous-time safety specification is $0.5 \leq t \leq 1$, $\theta \in [0, 1]$.

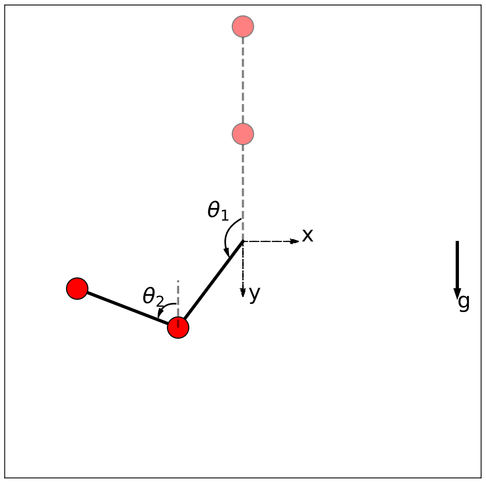

Figure 3: Inverted double pendulum. The goal is to keep the pendulum upright (dashed schematics)

3.6 Double Pendulum Benchmark

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:

$$2\ddot{\theta}_1 + \ddot{\theta}_2 \cos(\theta_2 - \theta_1) - \dot{\theta}_2^2 \sin(\theta_2 - \theta_1) - 2\frac{g}{L}\sin\theta_1 + \frac{c}{mL^2}\dot{\theta}_1 = \frac{1}{mL^2}T_1 \tag{8a}$$

$$\ddot{\theta}_1 \cos(\theta_2 - \theta_1) + \ddot{\theta}_2 + \dot{\theta}_1^2 \sin(\theta_2 - \theta_1) - \frac{g}{L}\sin\theta_2 + \frac{c}{mL^2}\dot{\theta}_2 = \frac{1}{mL^2}T_2 \tag{8b}$$

where θ_1 and θ_2 are the angles that links make with the upward vertical axis (seeFigure 3). The state is:

$$[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2] \tag{9}$$

The angular velocity and acceleration of links are denoted with $\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1$ and $\ddot{\theta}_2$ and g is the gravitational acceleration.

The controllers for the double pendulum benchmark are obtained using the same methods as the controllers for the single pendulum benchmark: behavior cloning from LQR and iLQR trajectories. The continuous-time equations of motion may be written as a series of first order ODEs where $x_1 = \theta_1$, $x_2 = \theta_2$, $x_3 = \dot{\theta}_1$ and $x_4 = \dot{\theta}_2$. See eq 11. The difference equations for the discrete time version of the system are obtained by using forward Euler integration:

$$x_{i_{t+1}} = x_{i_t} + \dot{x}_{i_t}\Delta t \text{ for } i \in [1,2,3,4]. \tag{10}$$

The model involves several parameters:

$$m = 0.5, L = 0.5, c = 0., g = 1.0.$$

Specification This benchmark has two controllers, each with slightly different specifications. Use `controller double pendulum less robust` with $\Delta t = 0.05$. The initial set is

$$[\theta_1, \theta_2, \dot{\theta}_1\dot{\theta}_2] = [1.0, 1.3]^4.$$

The safety specification is

$$\forall n_t \leq 20, [\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] \in [-1.0, 1.7]^4.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = 0.05 * 20 = 1sec.$

Use `controller double pendulum more robust` with $\Delta t = 0.02$. The initial set is

$$[\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] = [1.0, 1.3]^4.$$

The safety specification is

$$\forall n_t \leq 20, [\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] \in [-0.5, 1.5]^4.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = 0.02 * 20 = 0.4sec.$

These equations are generated with Matlab:

$$\dot{x}_1 = x_3 \tag{11a}$$

$$\dot{x}_2 = x_4 \tag{11b}$$

$$\dot{x}_3 = \frac{\star \left({x_3}^2 \sin(x_1 - x_2) - \star \left(\frac{g \sin(x_1)}{L} - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{T_1 - c\,x_3}{2\,L^2\,m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\,x_4}{L^2\,m} \right)}{2 \left(\frac{\cos^2(x_1 - x_2)}{2} - 1 \right)} \tag{11c}$$

$$- \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{g \sin(x_1)}{L} + \frac{T_1 - c\,x_3}{2\,L^2\,m} \tag{11d}$$

$$\dot{x}_4 = - \frac{{x_3}^2 \sin(x_1 - x_2) - \star \left(\frac{g \sin(x_1)}{L} - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{T_1 - c\,x_3}{2\,L^2\,m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\,x_4}{L^2\,m}}{\frac{\cos^2(x_1 - x_2)}{2} - 1}, \tag{11e}$$

where $\star = \cos(x_1 - x_2)$.

3.7 Airplane Benchmark

The airplane example consists of a dynamical system that is a simple model of a flying airplane. It can be visualized in Figure 4. The state is:

$$[x, y, z, u, v, w, \phi, \theta, \psi, r, p, q] \tag{12}$$

where (x, y, z) is the position of the C.G., (u, v, w) 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:

$$\dot{u} = -g \sin\theta + \frac{F_x}{m} - qw + rv \tag{13a}$$

$$\dot{v} = g \cos\theta \sin\phi + \frac{F_y}{m} - ru + pw \tag{13b}$$

$$\dot{w} = g \cos\theta \cos\phi + \frac{F_z}{m} - pv + qu \tag{13c}$$

$$I_x \dot{p} + I_{xz} \dot{r} = M_x - (I_z - I_y) qr - I_{xz} pq \tag{13d}$$

$$I_y \dot{q} = M_y - I_{xz} \left(r^2 - p^2 \right) - (I_x - I_z) pr \tag{13e}$$

$$I_{xz} \dot{p} + I_z \dot{r} = M_z - (I_y - I_x) qp - I_{xz} rq. \tag{13f}$$

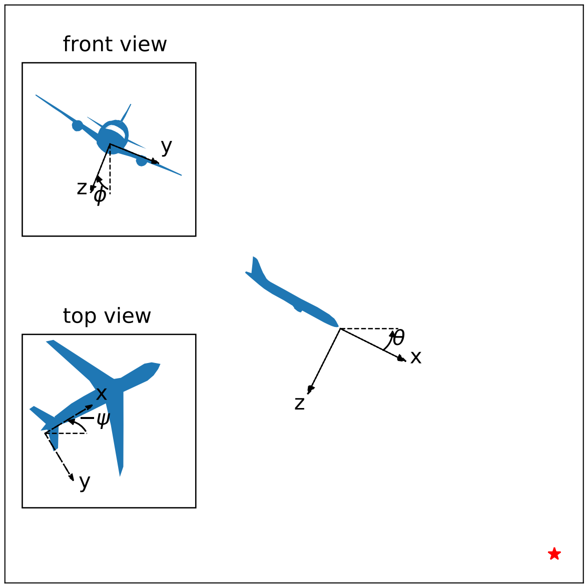

Figure 4: The airplane example.

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 controls parameters include three force components F_x, F_y and F_z and three moment components M_x, M_y, M_z. Notice that for simplicity, we have assumed the aerodynamic forces are absorbed in the F's. In addition to these six equations, we have six additional kinematic equations:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{14}$$

and

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}1 & \tan\theta\sin\phi & \tan\theta\cos\phi\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{15}$$

As in the pendulum benchmarks, controllers are trained for the airplane problem using behavior cloning from LQR and iLQR trajectories.

The state is defined to be $[x, y, z, u, v, w, \phi, \theta, \psi, r, p, q]$ and the derivatives that form the system of continuous time equations are specified in eqs 13,14,15. The difference equations for the discrete time version of the system are obtained by using the following mapping: $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}] = [x, y, z, u, v, w, \phi, \theta, \psi, r, p, q]$ and forward Euler integration:

$$x_{i_{t+1}} = x_{i_t} + \dot{x}_{i_t}\Delta t \text{ for } i \in [1, 2, 3, 4]. \tag{16}$$

The system involves the following model parameters:

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

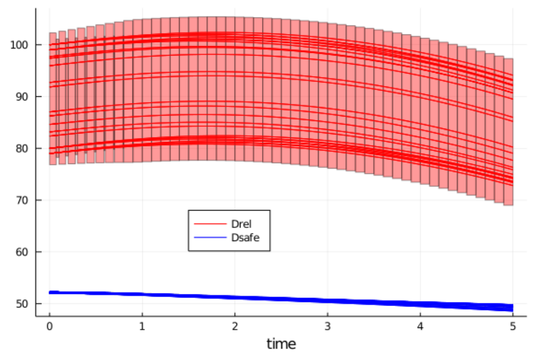

Figure 5: ***JuliaReach.*** Analysis results of the ACC benchmark. The plot additionally shows simulations.

Use the `controller airplane` with $\Delta t = 0.1$. The initial set is

$$x = y = z = r = p = q = 0, [u, v, w, \phi, \theta, \psi] = [0.0, 1.0]^6.$$

Specification The safety specification is

$$\forall n_t : n_t \leq 20, y \in [-0.5, 0.5], [\phi, \theta, \psi] = [-1.0, 1.0]^3.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = .1 * 20 = 2sec$.

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 seven benchmark problems (three verified, four falsified). For each problem, JuliaReach uses slightly different settings and sometimes only analyzes a certain scenario, 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 half a second. The reach sets of D_{rel} and D_{safe} together with some simulations are shown in Figure 5.

4.1.2 Sherlock-Benchmark-9

The TORA benchmark problem is challenging because the uncertainty in the variables x_3 and x_4 grows quickly in a set-based analysis. Using the parameters `abstol=1e-10, orderT=8, orderQ=3` and splitting the initial states into $4 \times 4 \times 3 \times 5$ boxes, JuliaReach verifies the property in 34 minutes for the homogeneous controller. 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.

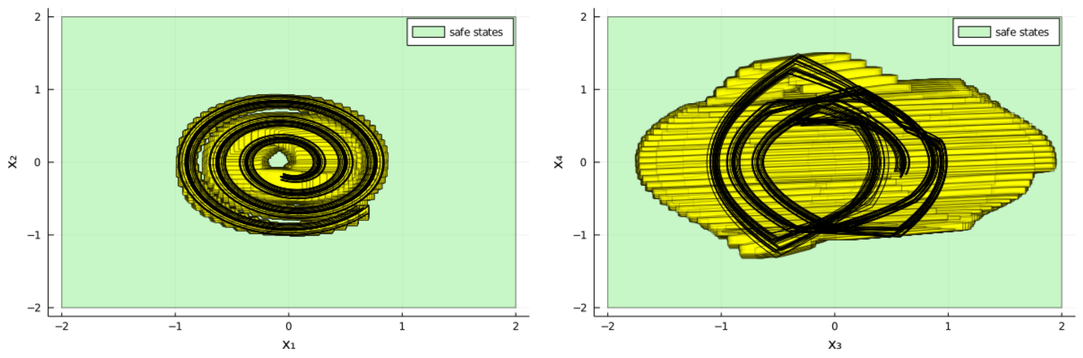

Figure 6: ***JuliaReach.*** Analysis results of the TORA benchmark. The plots additionally show simulations.

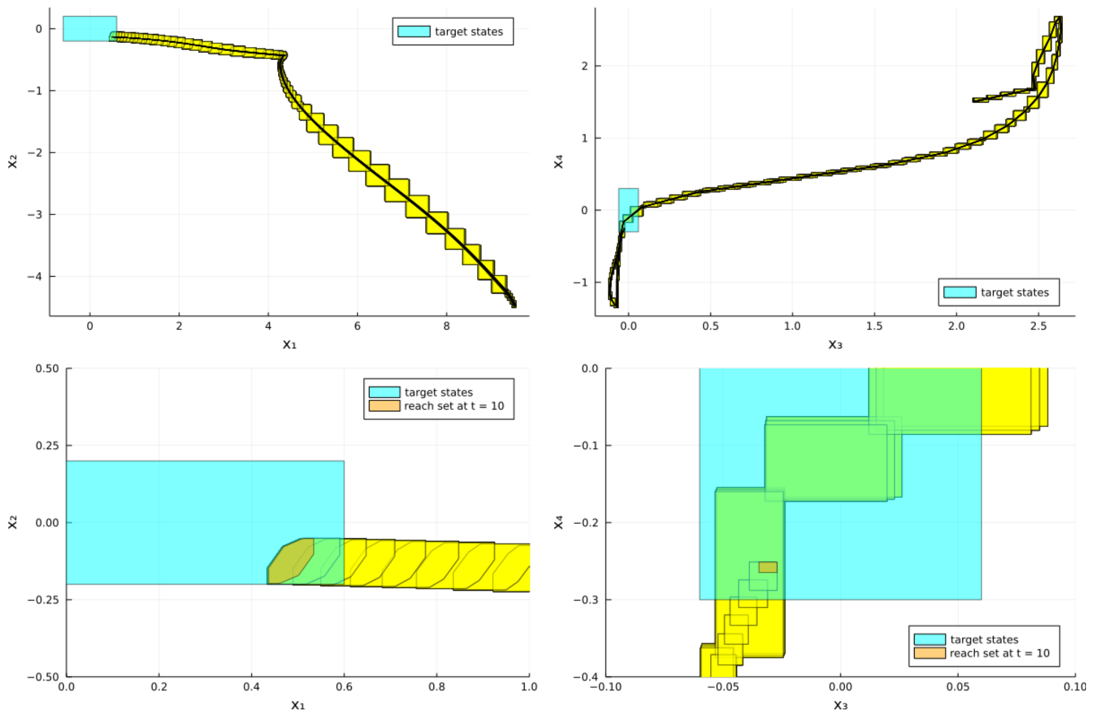

Figure 7: ***JuliaReach.*** Analysis results of the Unicycle benchmark. The first two plots show the overall reach sets and simulations. The other two plots show a close-up of the target set. The orange subset of the last reach set is obtained at time point $t = 10$.

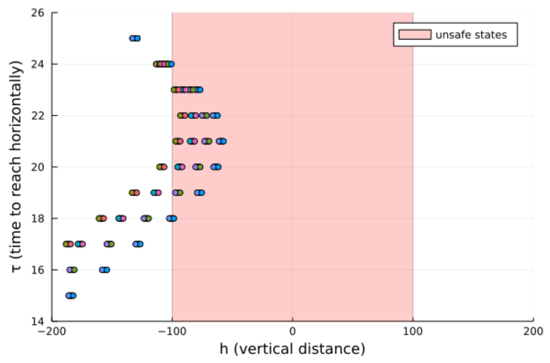

Figure 8: ***JuliaReach.*** Simulations of the VCAS benchmark.

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 100 seconds. 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 for this problem that additional precision is not required: the reach set for the last time interval is already fully contained in the target set.

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 and falsifies the property in one second. 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, when 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 half a second. The reach sets together with a simulation, projected to time and θ, are shown in Figure 9.

4.1.6 Double Pendulum

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. Here, when considering the less robust controller and starting from the highest coordinate in each dimension, a violation occurs within five control periods. Similarly, when considering the more robust

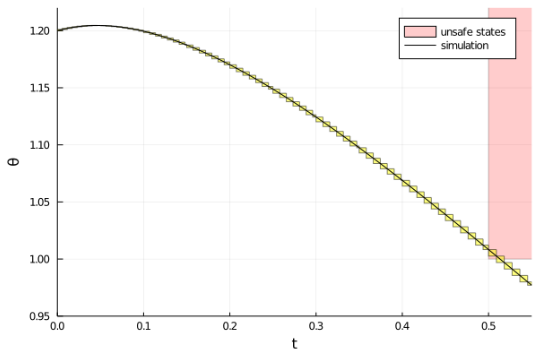

Figure 9: ***JuliaReach.*** Analysis results of the Single-Pendulum benchmark until time $t = 0.55$. The plot additionally shows a simulation.

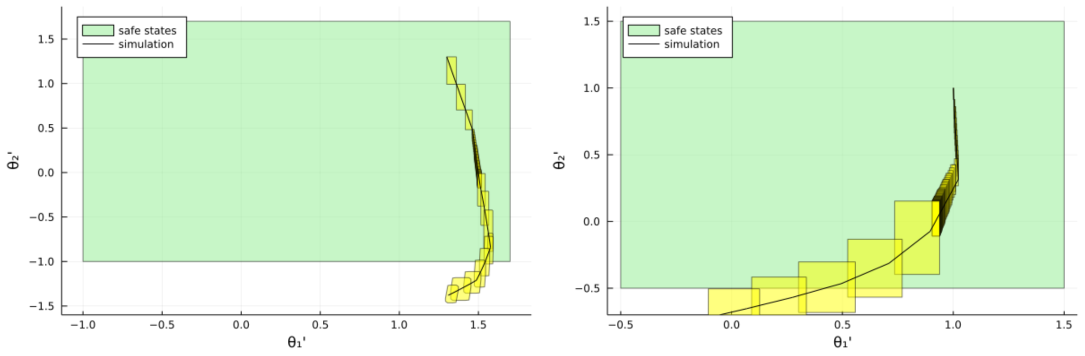

Figure 10: ***JuliaReach.*** Analysis results of the Double-Pendulum benchmark. The first plot shows the results for the less robust controller until time $t = 0.25$. The second plot shows the results for the more robust controller until time $t = 0.14$. The plots additionally show a simulation.

controller and starting from the lowest coordinate 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 46 seconds (less robust controller) resp. 40 seconds (more robust controller). The reach sets together with a simulation, projected to $\dot{\theta}_1/\dot{\theta}_2$, are shown in Figure 10.

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. Here, 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 nine seconds. The reach sets together with a simulation, projected to y/ϕ resp. θ/ψ, are shown in Figure 11.

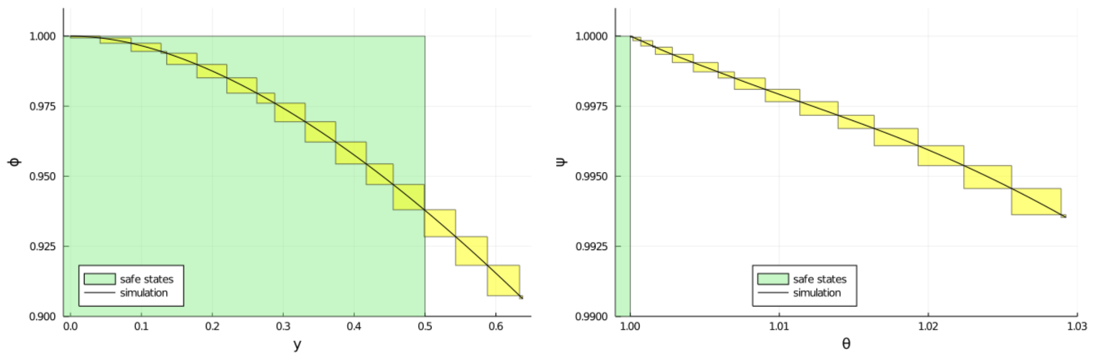

Figure 11: ***JuliaReach.*** Analysis results of the Airplane benchmark until time $t = 0.4$. The plots additionally show a simulation.

4.2 NNV

We present the results utilizing *NNV* on each of the benchmarks. One addition to this year's results is the comparison between two reachability methods used for continuous-time nonlinear systems. The first one, referred to as *zono*, uses zonotopes to compute the reach sets of the plants and star sets for the controller, while the second one, referred to as *poly*, uses polynomial zonotopes to compute the reach sets of the plant and star sets for the controller. For details about the reachability parameters used, such as reachability step size for continuous-time systems or the zonotope order, we refer to the submission code available at NNV/ARCH2021.

4.2.1 ACC

For the ACC benchmark, we present results using a neural network controller with 101 neurons (5-by-20) with ReLU activation functions, and use a time horizon of 5 seconds, as shown in Figure 12. We observe that the safety of the car is guaranteed, as the intersection of the actual (relative) and safe distance is empty throughout the 5-second time horizon. Results are only shown using the *zono* method.

4.2.2 Sherlock-Benchmark-9

The results are displayed in Figure 13, being the *lin* method on top and the *poly* on the bottom. The result is unknown due to the use of over-approximation analysis. Although the first three dimensions satisfy the safety property, the fourth does not on both executions.

4.2.3 Sherlock-Benchmark-10

On this benchmark, NNV runs out of memory due to the number of computation steps and splits needed in the computation of the plant's reachable sets. We have plotted a graph with the first two dimensions in Figure 14. One can see how the size of the reachable sets increases as time progresses. This growth leads to NNV running out of memory with both methods.

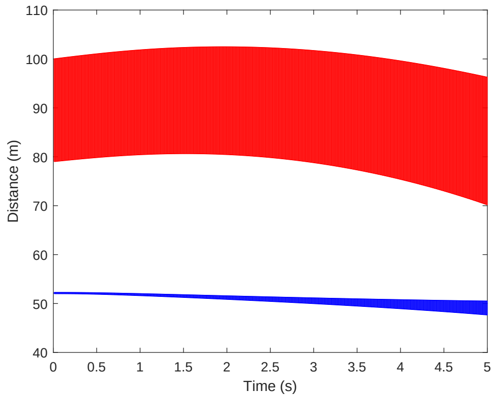

Figure 12: ***NNV.*** Reachability analysis results of the ACC benchmark using a controller with 5 hidden layers (ReLU) of 20 neurons each.

4.2.4 TORA Heterogeneous

The experimental results for both of the controllers considered in our analysis are similar. Both result in a verification result of unknown due to the over-approximate scheme utilized for nonlinear functions. In both cases, we initialize the analysis with a smaller initial set than initially proposed in order to promote faster computation. The experiments demonstrate that due to the over-approximation scheme, the system may reach the proposed area, but it is not guaranteed. The results for the ReluTanh controller are shown in Figure 15 and the results for the sigmoid controller are shown in Figure 16. The *zono* results are on the left and the *poly* results on the right, where only the reach sets at each control period are shown.

4.2.5 VCAS

For the VCAS Benchmark, there are two scenarios that we investigate. The first one is the worst-case scenario, where we show that after 3 steps, the system is unsafe under all possible initial velocities. The results are shown in Figure 17. For the second case, choosing the middle acceleration when possible, we also show that the system is unsafe after a few steps for all possible initial velocities. Results are shown in Figure 18.

4.2.6 Single Pendulum

The verification result for the single pendulum system is unsafe, as shown in Figure 19. The angle θ is not within the bounds [0.0, 1.0] after 10 time steps in both executions.

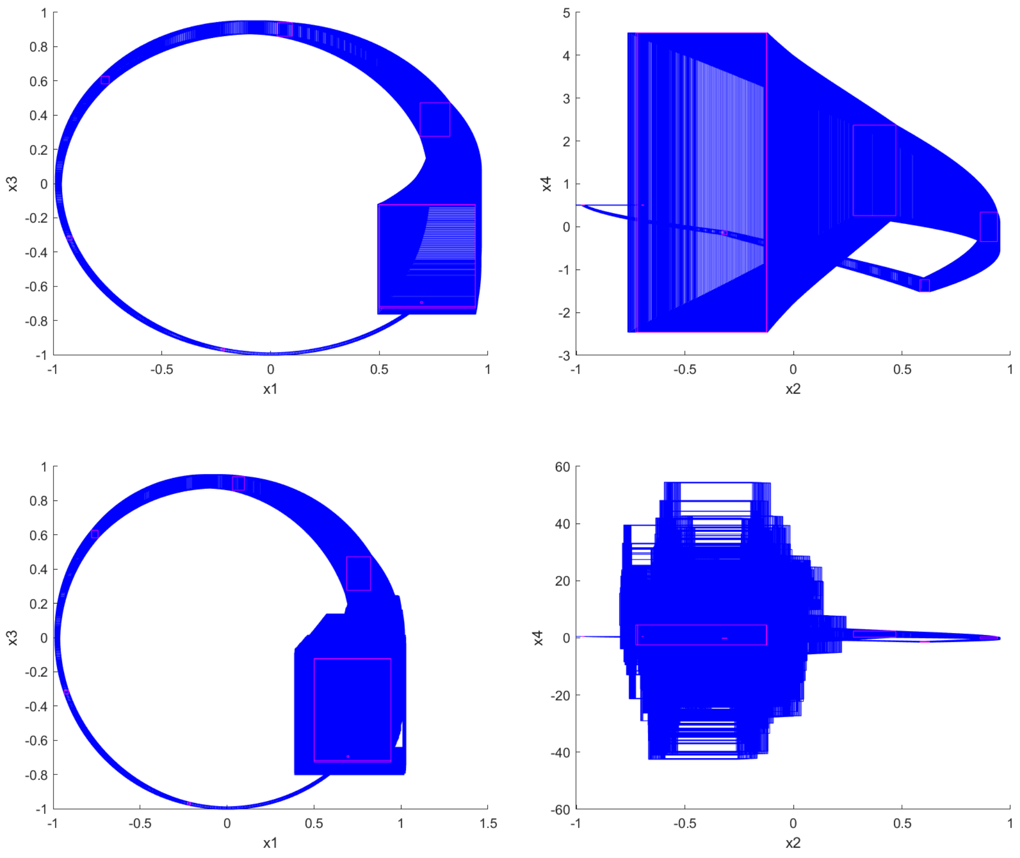

Figure 13: ***NNV.*** Reachability analysis results of the TORA Sherlock benchmark 9. All dimensions are shown: Dimensions 1 and 2 are plotted in the left figure, dimensions 3 and 4 in the right. *zono* on the top, *poly* on the bottom.

4.2.7 Double Pendulum

The results for the system with the less robust controller are shown in Figure 20. One can see that dimensions 1, 2 and 4 still satisfy the safety property, but dimension 3 does not due to the over-approximate analysis. The results for the system with the more robust controller are shown in Figure 21. This is a similar case to the previous controller, as dimension 3 does not satisfy the safety property while the other still do. Both plots present the *zono* results on top and the *poly* results on the bottom.

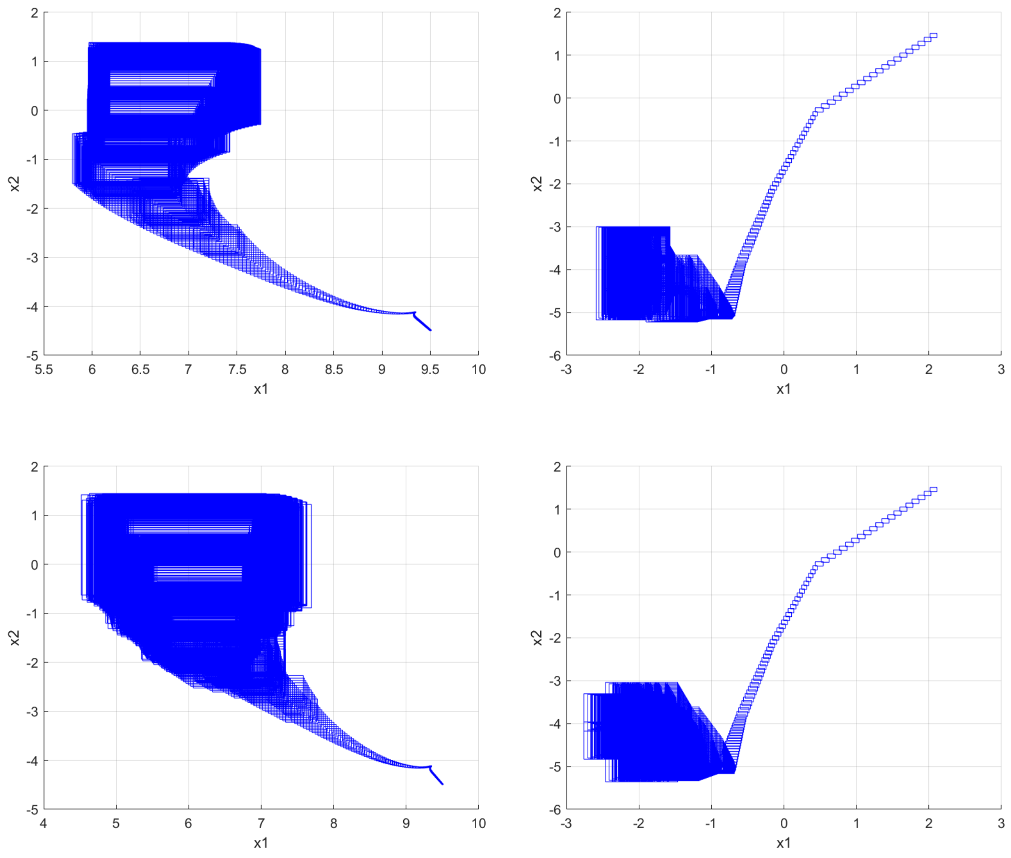

Figure 14: ***NNV.*** Reachability Analysis results for benchmark 10 (Unicycle). *zono* method shown on top, *poly* on the bottom.

4.2.8 Airplane

The airplane benchmark is a high-dimensional nonlinear benchmark, which makes the analysis very computationally expensive. The analysis was computed for 13 control steps. After these steps, each dimension satisfies the safety property except for dimension 2, which due to the over-approximation in the reachable sets, reaches the -1 and 1 safety boundaries both the *zono* and *poly*. The resulting reachable sets are shown in Figure 22.

4.3 Verisig

Verisig supports NNs with smooth activations, so it only applies to two of the benchmarks in this year's competition.

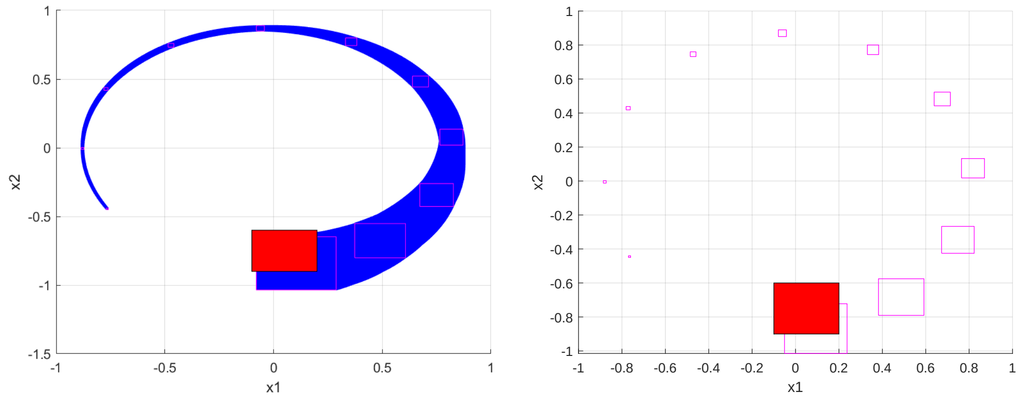

Figure 15: ***NNV.*** Reachability analysis results of TORA with ReLU-Tanh controller using *zono* on the left and *poly* on the right.

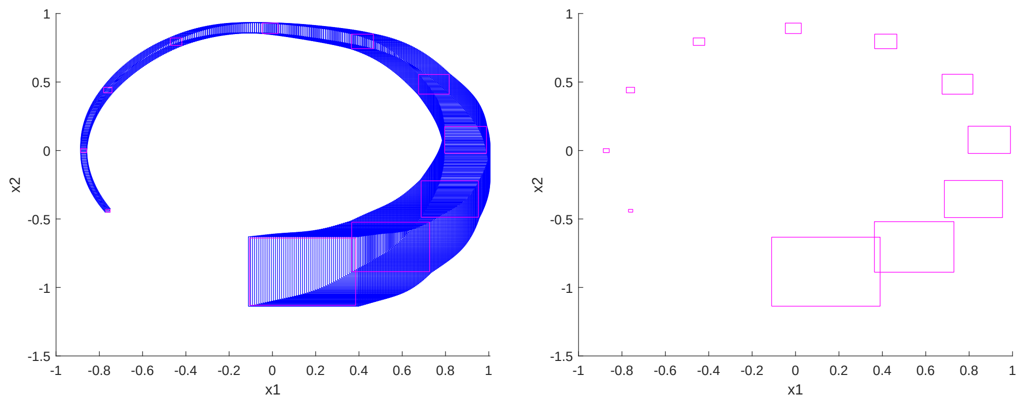

Figure 16: ***NNV.*** Reachability analysis results of TORA with sigmoid controller using *zono* on the left and *poly* on the right.

4.3.1 ACC

We used Verisig on the ACC benchmark with a tanh controller. Since that controller results in slightly different performance, we modified the initial conditions to the following set: $x_{\text{lead}}(0)$ $\in$ [90,91], $v_{\text{lead}}(0) \in$ [32,30.05], $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.05], $x_{\text{ego}} \in$ [10,11]. The reachable sets for $Drel$ and $Dsafe$ are shown in Figure 23. As can seen in the figure, the safety of the car is guaranteed for the 5s window of interest.

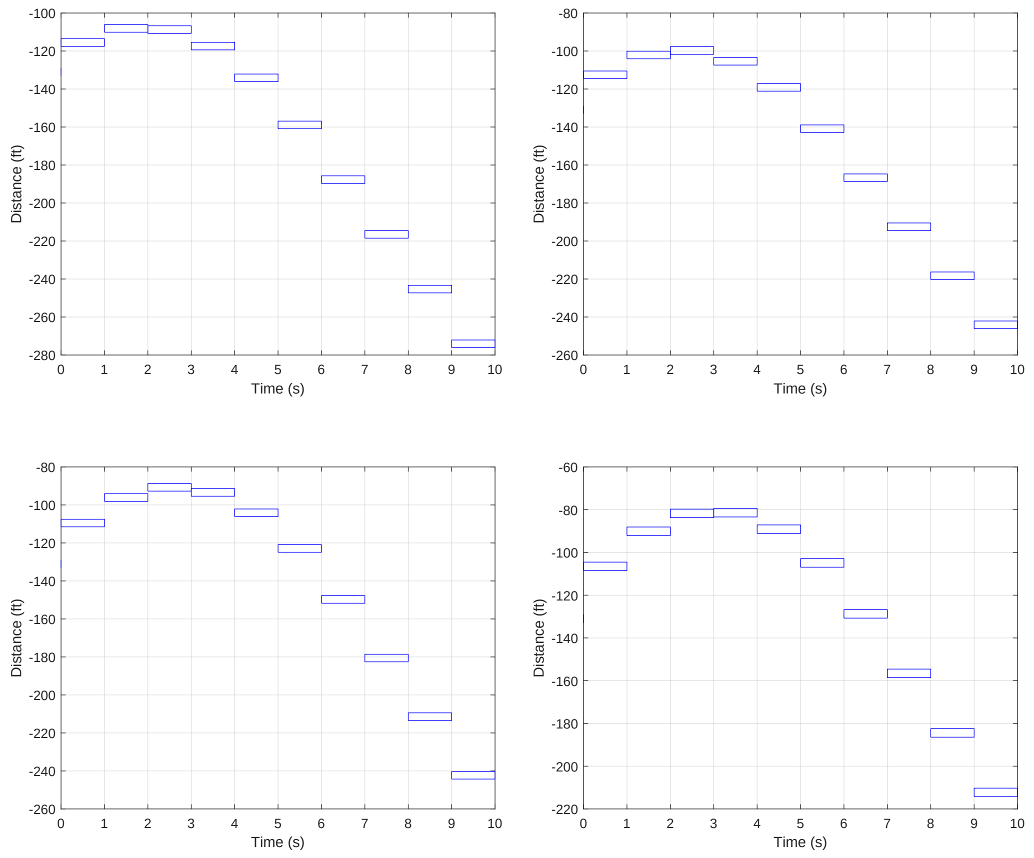

Figure 17: ***NNV.*** VCAS results when choosing the worst possible acceleration.

4.3.2 Tora Heterogeneous

The reachable sets produced by Verisig on the Tora benchmark with a sigmoid controller are presented in Figure 24. As can be seen in the figure, Verisig is able to maintain tight reachable sets due to effective preconditioning and shrink wrapping.

5 Category Status and Challenges

In the third iteration of the AINNCS category at ARCH-COMP, the participating tools JuliaReach, NNV and Verisig successfully analyzed different aspects of the benchmark problems. In spite of some success analyzing the benchmarks, the primary outcome of this third iteration of the AINNCS category are the challenges that arose in the competition and the improvements with respect to the last two competitions. We discuss these next.

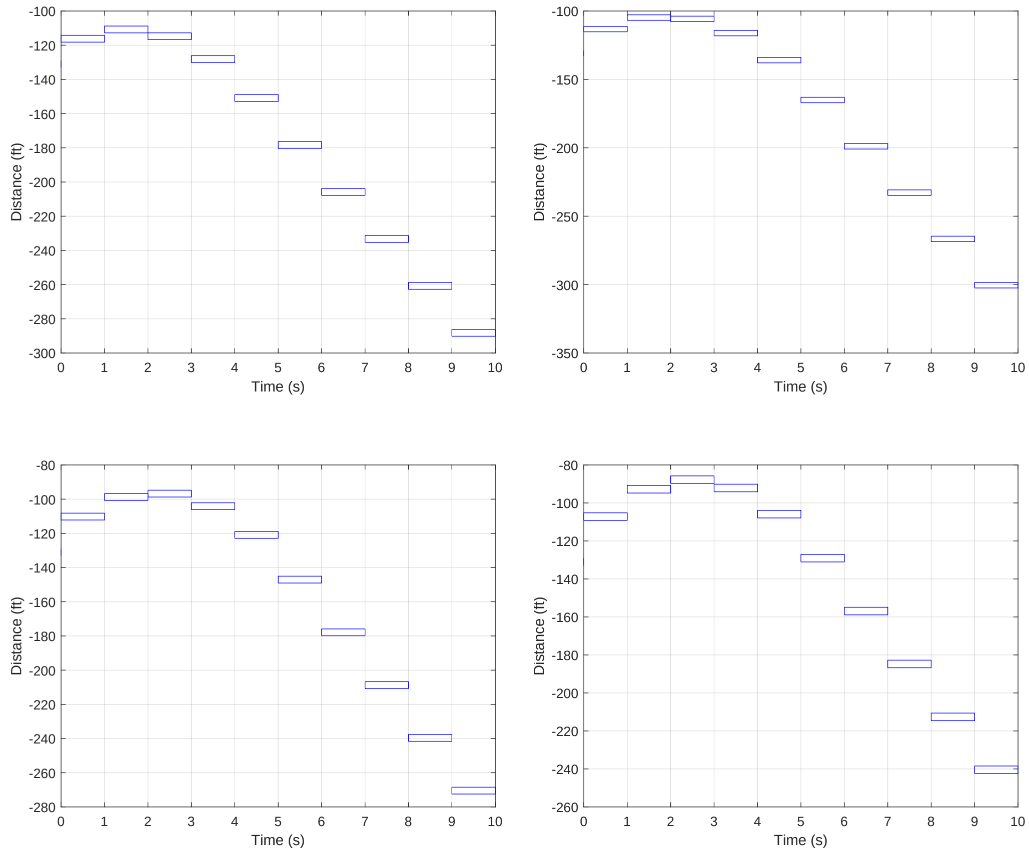

Figure 18: ***NNV.*** VCAS results when we choose the middle acceleration.

Hybrid Controllers: Some controllers involve a hybrid nature. In this year's competition, this type of controller only appears in 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 two thirds of the tools successfully verify this benchmark, which is an improvement from previous iterations.

Plant Models: This year we have only considered nonlinear plants, both in discrete and continuous time. A majority of the tools only support discrete or continuous time, with Verisig being the only tool with no support for both type of dynamics (only continuous-time). In comparison to last year, only 1 out of the 4 participating tools had direct support for both kind of dynamics. We plan to add linear as well as 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.

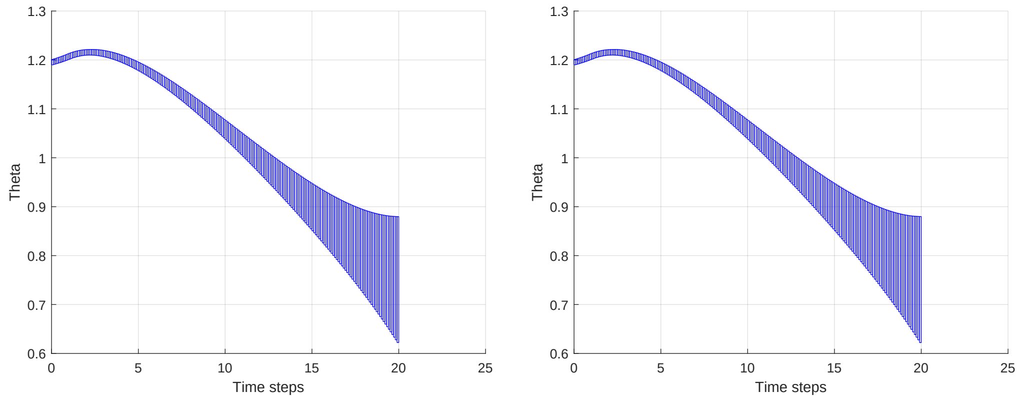

Figure 19: ***NNV.*** Single pendulum reachable sets. On the left, *zono*, on the right, *poly*.

Activation Function Types: 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. This is a step forward from last year's competition, as nonlinear activation functions were included. However, not all tools have support for all activation functions, with some supporting only ReLU and linear activation functions at the time the report was written.

Neural Network Architectures and Parameterization: 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. Thus, the dimensionality of the neural network controllers and plant states have significantly increased compared to first year. We plan to increase the complexity of these in the upcoming years. Additionally, the verification results presented assumed the neural networks are fixed, while other parameterizations are possible, some of which were partly explored (e.g., using different network architectures or activation functions for a given plant). In any event, there are state-space explosion and scalability issues to address in both the neural network controllers and plant analysis.

Time horizons: Similar to last year's competition, 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 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

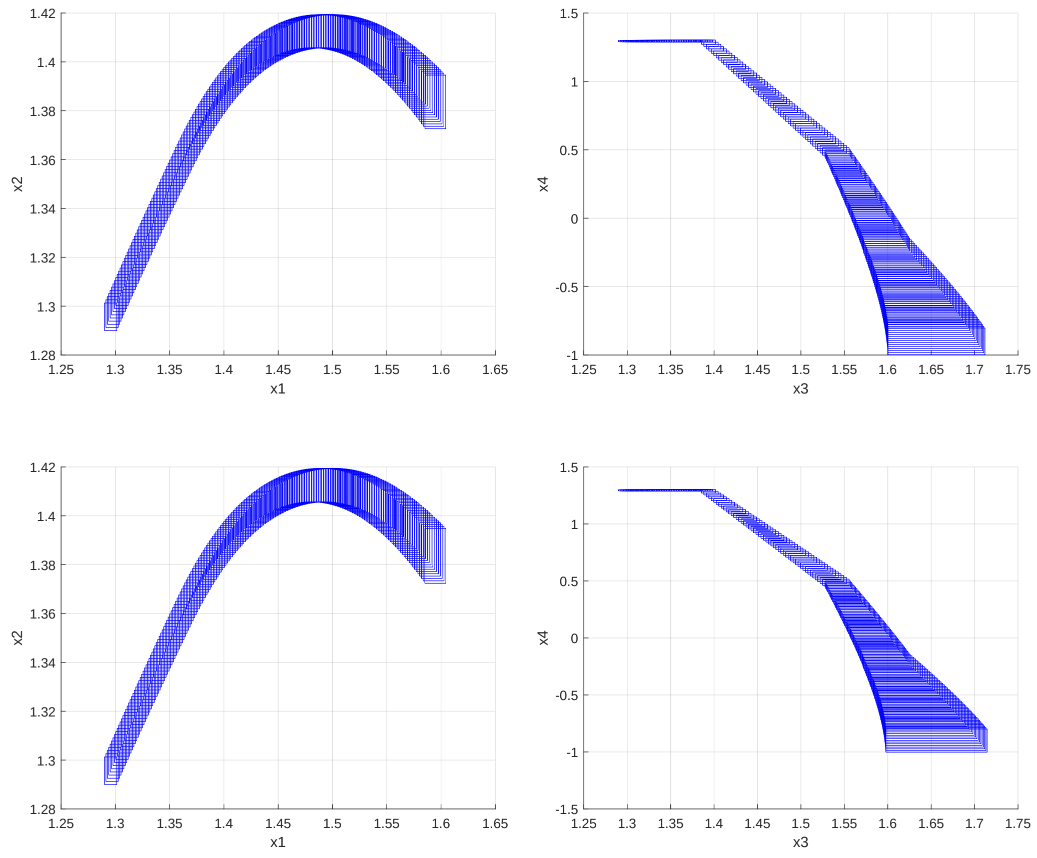

Figure 20: ***NNV.*** Double pendulum with less robust controller.

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 discrepancies among the different versions and frameworks these ONNX models were created from. Having a unified ONNX version remains a challenge, but we are closer to achieving this goal, and initiatives

[3]Open Neural Network Exchange: https://github.com/onnx/onnx
[4]Direct input format used by *NNV* without transformation.

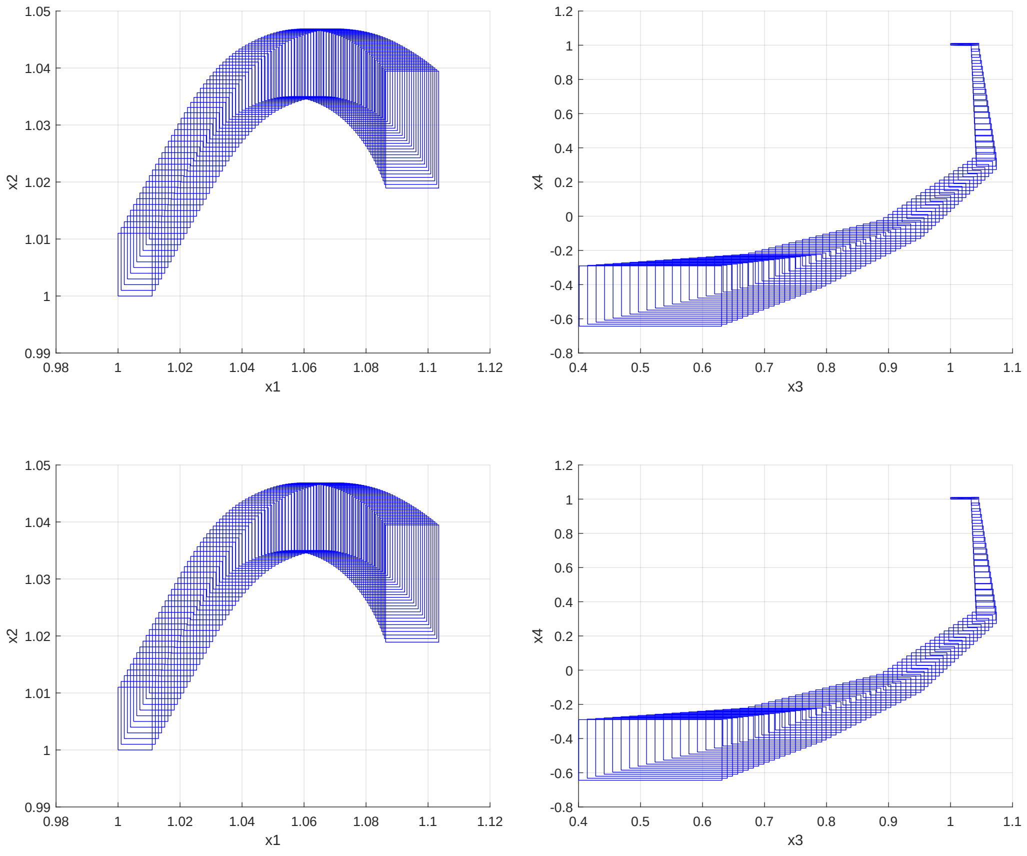

Figure 21: ***NNV.*** Double pendulum with more robust controller.

more focused on neural network verification, such as VNN-LIB[5] and VNN-COMP[6], may help toward this goal. On the other hand, we have found some disparities in the understanding of some specifications / normalization variables in two benchmarks. In future competitions, we will further discuss other input formats for specifications and normalization functions to eliminate these misinterpretations.

6 Conclusion and Outlook

This report presents the results on the third ARCH friendly competition for closed-loop systems with neural network controllers. For the third edition, three tools have participated and attempted to solve 7 benchmarks: JuliaReach, NNV, and Verisig. 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. We note that each tool has unique strengths and that not all of the specificities can be highlighted within a single report.

[5]http://www.vnnlib.org/
[6]https://github.com/verivital/vnn-comp/

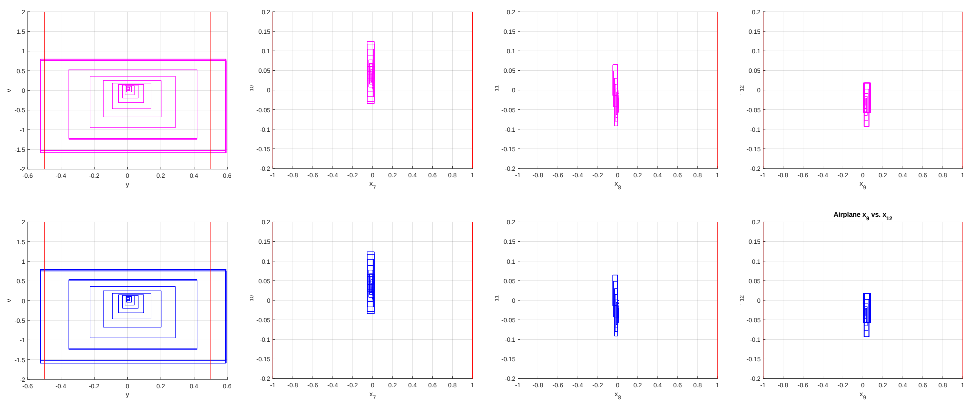

Figure 22: ***NNV.*** Airplane reachable sets. In red, the safety boundaries. On the top, the *zono* results (magenta) and *poly* results on the bottom (blue).

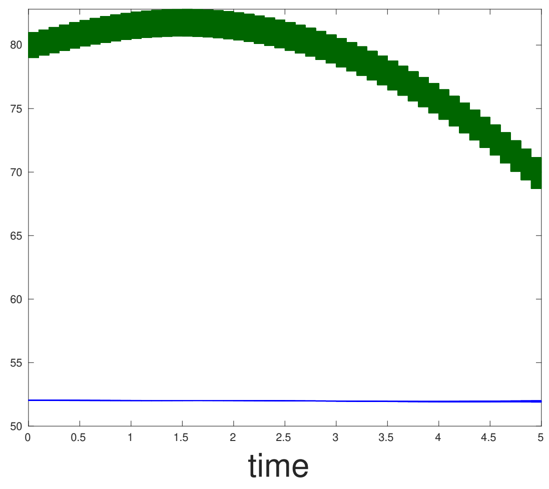

Figure 23: Reachable sets produced by Verisig on the ACC benchmark with a tanh controller. Reachable sets for $Drel$ and $Dsafe$ are shown in green and blue, respectively.

However, 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 two years, we observe that the numbers of benchmarks and difficulty of these have increased from the first iteration, which is a good indicator for this growing and maturing field. We have also seen some improvements in some of the tools that previously participated in the first iteration, and new tools being developed every year, such as the recent JuliaReach that help push this field forward.

We would also like to encourage other tool developers to consider participating next year. All authors agree that although the participation consumes time, we have gained unique insights that will allow us to improve in the next iteration. Particularly those items listed in the status and challenges section. The reports of other categories can be found in the proceedings and on the ARCH website: cps-vo.org/group/ARCH.

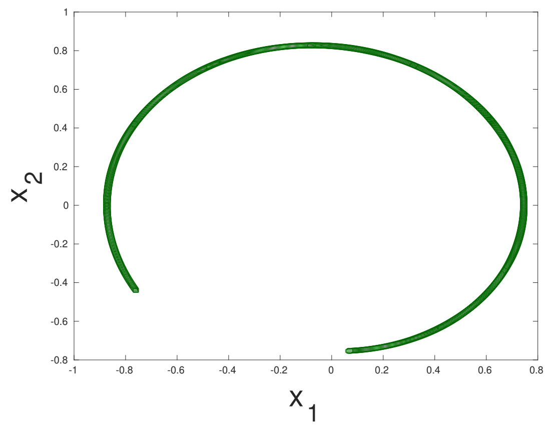

Figure 24: Reachable sets produced by Verisig on the Tora Heterogeneous benchmark with a sigmoid controller.

7 Acknowledgments

The material presented in this report is based upon work supported by the National Science Foundation (NSF) under grant numbers FMitF 1918450 and EPCN 2028001, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any 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 DARPA or NSF. Luis Benet acknowledges support from PAPIIT grant IG-100819.

A Specification of Used Machines

A.1 M$_{JuliaReach}$

- Processor: Intel Core i5-5200U CPU@2.20GHz, running 64-bit Ubuntu 20.04.
- Memory: 8 GB

A.2 M$_{nnv}$

- Processor: Intel Core i7-8750H CPU @ 2.20GHz x 12
- Memory: 32 GB

A.3 M$_{verisig}$

- Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz x 40
- Memory: 755 GB

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] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] 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.

[14] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[15] 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.

[16] 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.

[17] 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.

[18] Marcelo Forets and Christian Schilling. JuliaReach/LazySets.jl. `https://github.com/JuliaReach/LazySets.jl`, 2021.

[19] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Case study: Verifying the safety of an autonomous racing car with a neural network controller. In *International Conference on Hybrid Systems: Computation and Control*, 2020.

[20] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *33rd International Conference on Computer-Aided Verification*, 2021.

[21] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control*, 2019.

[22] Radoslav Ivanov, Taylor J. Carpenter, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verifying the safety of autonomous systems with neural network controllers. *ACM Trans. Embed. Comput. Syst.*, 20(1), December 2020.

[23] 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.

[24] 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.

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

[26] 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.

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

[28] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher A. Strong, Clark W. Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Found. Trends Optim.*, 4(3-4):244–404, 2021.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.