Safety Verification of Neural Network Controlled Systems

Arthur Clavière, Eric Asselin, Christophe Garion, Claire Pagetti

Safety Verification of Neural Network Controlled Systems

Arthur Clavière
Collins Aerospace
Toulouse, France

Eric Asselin
Collins Aerospace
Toulouse, France

Christophe Garion
ISAE-SUPAERO
Toulouse, France

Claire Pagetti
ONERA
Toulouse, France

***Abstract*—In this paper, we propose a system-level approach for verifying the safety of systems combining a continuous-time physical system with a discrete-time neural network based controller. We define a generic modelling approach and an associated reachability analysis that soundly approximates the reachable states of the overall system. We illustrate our approach through a real-world use case.**

***Keywords*-Neural Networks, Formal Methods, Certification**

I. Introduction

A. Context and contribution

Recently, feedforward deep neural networks have been successfully used for controlling physical systems, such as self-driving cars [BTD+16], [CSKX15], [PCS+17] and unmanned aerial vehicles [JKO18]. The combination of a physical system with a neural network based controller is sometimes known as a *neural network controlled system*. If such a system is considered as *safety-critical*, meaning that a failure of the system could have serious consequences, then a particular effort needs to be made to demonstrate its safety. To do so, usually, the system has to be developped in accordance with stringent standards *e.g.*, ED-79A/ARP-4754A [EUR10] in aeronautics. In particular, the system requirements have to be refined at the *item level*, with the aim of achieving a *correct*, *comprehensive* specification for each item composing the system. Then, the development of each item must be performed in compliance with dedicated standards, *e.g.*, ED-12C/DO-178C [EUR11] in aeronautics, which prescribes several verification activities to prove that a software item behaves *exactly* as expected.

This classical approach is not applicable to neural network controlled systems. First, one cannot refine the system requirements at the neural network level as, most of the time, one cannot achieve a correct, comprehensive specification for the expected behaviour. Generally, the expected behaviour of a network consists of a set of example data, which is a *pointwise, non-comprehensive* specification. Secondly, existing standards such as ED-12C/DO-178C are not applicable as, even if a comprehensive specification could be defined, the learning process does not guarantee the correcteness of the resulting network. As a consequence, verifying that a network behaves exactly as expected may be infeasible, precisely because it does not.

This paper proposes a *generic* approach for demonstrating the safety of a particular class of neural network controlled systems, where the controller is a *classifier* based on *multiple* ReLU networks. To tackle the above mentioned issues, our approach consists of a *system-level* approach that provides evidence that the overall system is safe, without performing item-level refinements and analyses. To this end, we leverage an accurate model of the overall system and we perform a reachability analysis to formally demonstrate that no reachable state can lead to a failure of the system. We make an evaluation of the approach *applicability* on real-world use cases and a *comparative study* with other existing tools.

The paper is organized as follows. Section II presents related works. Section III describes our model of the targetted class of neural network controlled system. Section IV details our reachability analysis for solving the verification problem and section V presents some experimental results. The ACAS Xu system case will serve as an illustration and the goal is to show that the controller is effectively safe *i.e.*, it does prevent near mid-air collision. A longer version of the paper is available [CAGP20].

B. Example use case

The safe integration of Unmanned Aerial Vehicles (UAVs) into the air traffic requires them to have collision avoidance capabilities. For this purpose, the standardization group RTCA SC 147 [EUR20] has recently developped a dedicated controller, namely the Airborne Collision Avoidance System for Unmanned Aircraft (ACAS Xu). The role of ACAS Xu is to avoid any collision between the *ownship*, equipped with the controller, and an encountered aircraft called the *intruder*, equipped or not with the controller. To this end, the ACAS Xu periodically provides the ownship with a *horizontal maneuver advisory*, being either clear-of-conflict (COC), weak left turn (WL), weak right turn (WR), strong left turn (SL) or strong right turn (SR). The optimal advisory is extracted from a set of lookup tables, depending on the previous advisory and six variables describing the encounter between the two aircraft, defined in Fig. 1: (1) the distance ρ from ownship to intruder, (2) the angle θ to intruder relative to ownship heading direction, (3) the heading angle ψ of intruder relative to ownship heading direction, (4) the velocity v_{own} of ownship, (5) the velocity v_{int} of intruder and (6) the time t_{sep} until loss of vertical separation. These six variables are computed from the input signals from the transponder and the sensors of the ownship *e.g.*, air-to-air radar, electro-optics/infrared sensors.

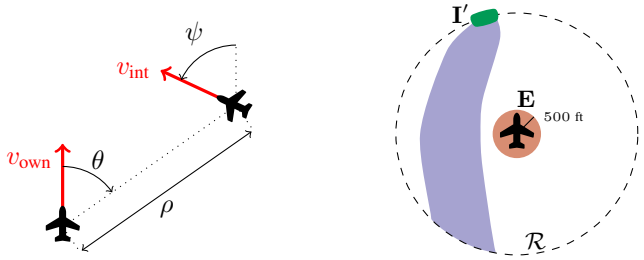


Fig. 1. The 2D geometry of the encounter between the two aircraft (left) and the (illustrative) reachable trajectories of intruder relative to ownship from a subset $\mathbf{I}'$ of the possible initial states, with $\mathbf{E}$ representing a collision cylinder around the ownship and $\mathcal{R}$ delimiting the range of the ownship sensors (right).

The main weakness of the ACAS Xu controller is the large storage requirements for the tables (over 2GB). Recently, an alternative design for the ACAS Xu controller has been proposed [JKO18], with dramatically reduced memory footprint (about 3 MB). It consists of a collection of 45 neural networks approximating the lookup tables. Each single network approximates a table corresponding to a fixed previous advisory and a given interval for t_{sep}. Due to the complexity of the controller, we lack a proof that no collision can happen, whatever the initial state of the two aircraft (see Fig. 1).

II. Related work

Neural network level. In the past few years, some progress has been made towards a more comprehensive specification for the expected behaviour of a neural network. Indeed, several research works have identified *local* expected behaviours contributing to the overall expected behaviour of the network. Typically, a local behaviour consists of a pre-condition about the input of the network together with a post-condition about its output. An example of such a property is *adversarial robustness* (also called local robustness) which captures the capability of the network to react correctly to a slight perturbation of a *given* input [KBD+17], [HKR+20]. In recent years, there has been significant interest in verifying neural networks against this type of property, which has been shown to be a NP-complete problem [KBD+17]. Several dedicated *formal methods* have been proposed, with the advantage of providing a *sound* analysis, meaning that the network is said correct only if it is actually correct. Some of these specialized formal methods are based on Satisfiability Modulo Theory solving [KBD+17], [Ehl17], with the advantage of providing a *complete* analysis *i.e.*, the network is said incorrect only if it is actually incorrect. However, these methods are often expensive for large, real-world sized networks. In order to offer a more *scalable* analysis, other dedicated formal methods have been proposed, relying on abstract intepretation to soundly *approximate* the semantics of the network [WPW+18], [GMDC+18], [SGM+18], [SGPV19]. Yet, as they consist of an over-approximation, these methods do not provide a complete analysis.

As explained in the introduction, our work does not address the safety objectives at the neural network level but at the system level, so we do not seek to identify new local properties or to improve the existing techniques for verifying neural networks. However, we aim at using abstract interpretation based techniques to analyze the behaviour of the overall system. Indeed, such methods scale well to large networks and they provide not only a yes-or-no answer to a verification problem but also an approximation of the network semantics, that is helpful when reasoning about the overall system.

System level. Verifying the safety requirements at the system level, which corresponds to our approach, has been the object of a lot of insightful research. Indeed, there has been significant interest in verifying the safety of *hybrid systems*, exhibiting both continuous-time and discrete-time dynamics. Among the proposed methods, *falsification* aims at finding trajectories that violate a given safety property [BFG+19], [ALFS11]. Yet, even though falsification can prove that the system is unsafe, it cannot prove that the system is safe. *Reachability analysis* can provide such a proof of safety by constructing a *sound* approximation of the reachable states of the system and demonstrating that no reachable state can lead to a failure [CÁS13], [Alt15], [AdSC16]. However, these classical reachability methods are not directly applicable to neural network controlled systems, due to the hardness of characterizing the input-output mapping of a neural network. Very recently, in the same vein as this paper, some research works have addressed the problem of verifying the safety of neural network controlled systems [IWA+18], [DCS19], [HFL+19], [TYL+20]. These works all assume a physical system combined with a periodically-scheduled controller that is a *single* neural network *i.e.*, the input of the network is the sampled state of the physical system and the output of the network is the actuation command. To ensure the safety of such a system, they propose dedicated methods, all relying on reachability analysis. However, due to the switching mechanism between the networks, these methods are not applicable to the class of neural network controlled systems that we consider. Indeed, these methods cannot handle a controller that is a classifier based on multiple ReLU networks [LMT+19].

To the best of our knowledge, only two methods can handle the verification of the targetted class of systems: the method proposed by [ABKL20], relying on MILP programming, and the method proposed by [JK19], relying on reachability analysis. However, due to the use of MILP programming, [ABKL20] may not scale well to large systems. Similarly, as [JK19] computes the reachable states by exploring the entire state space, it may not scale to high-dimensional systems. Moreover, these two methods are not totally sound as they do not evaluate the reachable states for all instants but only for a set of *discrete* instants. We propose here a *scalable* approach for *soundly* verifying the safety of neural network controlled systems with a classifier based on multiple ReLU networks as a controller.

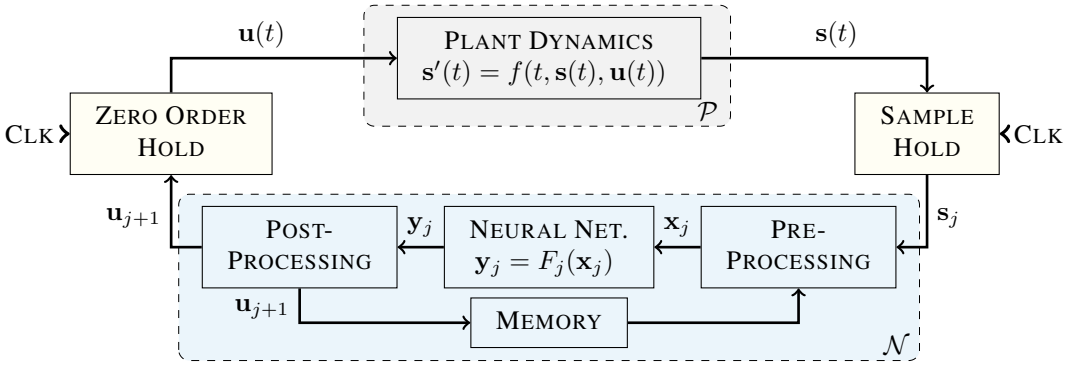


Fig. 2. Block diagram of a closed-loop system $\mathcal{C} = (\mathcal{P}, \mathcal{N})$.

III. System model

A. Closed-loop system

We assume a *closed-loop system* $\mathcal{C}$ that is the combination of a plant $\mathcal{P}$ and a controller $\mathcal{N}$ (see Fig. 2). The plant $\mathcal{P}$ is a *continuous-time* system, the state of which is the real-numbered vector $\mathbf{s}(t) \in \mathbb{R}^l$ at instant $t \in \mathbb{R}$. The evolution of $\mathbf{s}(t)$ is continuous and depends, inter alia, on the actuation command from the controller, denoted by $\mathbf{u}(t) \in \mathbb{R}^d$. The controller $\mathcal{N}$ is a *discrete-time* system, executed periodically with period T. The j^{th} execution of the controller (or control step) occurs in the time interval $[jT, (j+1)T[$. It takes as input the sampled state $\mathbf{s}_j = \mathbf{s}(jT)$ and it yields the command $\mathbf{u}_{j+1}$ to be applied for next period *i.e.*, $\mathbf{u}(t) = \mathbf{u}_{j+1}\ \forall t \in [(j+1)T, (j+2)T[$. It is worth noting that such a model does not assume the controller to execute instantaneously: its execution time only has to be less than T, as for real systems. Moreover, the controller is assumed to be a *classifier*, meaning that the command $\mathbf{u}_{j+1}$ produced by the controller is taken from a *finite* set $\mathbf{U} = \left\{\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(P)}\right\} \subset \mathbb{R}^d$, representing the possible actuation commands.

The plant and the controller interact by means of a signal sampler and a zero-order-hold. Overall, the state of the closed-loop system $\mathcal{C}$ is the 2-tuple $\phi(t) = (\mathbf{s}(t), \mathbf{u}(t))$ and we denote by $\phi_0 = (\mathbf{s}_0, \mathbf{u}_0) \in \mathbf{I}$ the initial state of $\mathcal{C}$, wherein $\mathbf{I} \subseteq \mathbb{R}^l \times \mathbf{U}$ is the set of the possible initial states.

Example 1: For the ACAS Xu system, we consider the plant $\mathcal{P}$ composed of both the ownship and the intruder. By assuming that the two aircraft are at the same altitude *i.e.*, $t_{\text{sep}} = 0$, we can define the state of $\mathcal{P}$ at instant t as the real-numbered vector $\mathbf{s}(t) = (x(t)\ y(t)\ \psi_{\text{own}}(t)\ \psi_{\text{int}}(t)\ v_{\text{own}}(t)\ v_{\text{int}}(t))^T$ where $x(t), y(t)$ are the 2D cartesian coordinates of intruder relative to ownship, $\psi_{\text{own}}(t)$ and $\psi_{\text{int}}(t)$ are the heading angles of ownship and intruder respectively (measured counter clockwise), $v_{\text{own}}(t)$ and $v_{\text{int}}(t)$ denote the velocities of ownship and intruder respectively (see Fig. 3). The controller $\mathcal{N}$ has a period $T = 1s$ and it outputs the actuation command $u(t) \in \mathbb{R}$ that is the turn rate of ownship, measured counter clockwise. This command is taken from the set $\mathbf{U} = \{0 \text{ deg/s}, 1.5 \text{ deg/s}, -1.5 \text{ deg/s}, 3 \text{ deg/s}, -3 \text{ deg/s}\}$, of which values represent COC, WL, WR, SL and SR respectively. Overall, an initial state $\phi_0 = (\mathbf{s}_0, u_0)$ of the closed-loop $\mathcal{C}$ corresponds to the intruder being detected by the ownship for the first time. Therefore, the initial position (x_0, y_0) of intruder lies along a circle $\mathcal{R}$ centered on ownship and with a radius r equal to the range of the ownship sensors (see Fig. 1). Furthermore, the initial heading of intruder $\psi_{\text{int},0}$ is such that the intruder penetrates the circle $\mathcal{R}$ *i.e.* $\psi_{\text{int},0}$ belongs to a cone delimited by the tangent to $\mathcal{R}$ at the point (x_0, y_0). The initial heading of ownship $\psi_{\text{own},0}$ can be taken equal to zero, without loss of generality, and the initial actuation command u_0 is 0.0 deg/s, corresponding to a Clear-of-Conflict.

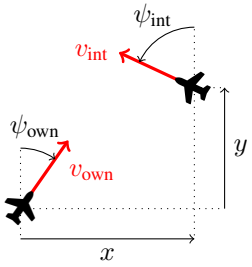
B. Plant dynamics

The dynamics of the plant $\mathcal{P}$ *i.e.*, the temporal evolution of its state $\mathbf{s}(t)$, is modelled by an *ordinary differential equation*.

Definition 1: An ordinary differential equation (ODE) is a relation between a function $\mathbf{z} : \mathbb{R} \to \mathbb{R}^l,\ t \mapsto \mathbf{z}(t)$ and its derivative $\mathbf{z}' = \frac{d\mathbf{z}}{dt}$ of the form $\mathbf{z}'(t) = f(t, \mathbf{z}(t))$ wherein $f : \mathbb{R} \times \mathbb{R}^l \to \mathbb{R}^l$.

To take account of the command signal $\mathbf{u}(t)$, the dynamics of $\mathcal{P}$ is of the form $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ wherein $f : \mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d \to \mathbb{R}^l$ is assumed to be continuous in t and $\mathbf{u}$ and uniformly Lipschitz continuous in $\mathbf{s}$ *i.e.*, its slope *w.r.t.* $\mathbf{s}$ is uniformly bounded on $\mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d$. Indeed, under these hypotheses and when $\mathbf{u}(t)$ is a piecewise constant function (as in the case of $\mathcal{C}$), then $\mathcal{P}$ has a deterministic behaviour. More precisely, let us consider a time interval $[0, qT]$ with $q \in \mathbb{N}$ and a given command signal $\mathbf{u}(t)$, constant on $[jT, (j+1)T[$ for $j < q$. There exists a unique function $\mathbf{s}^*$ defined on $[0, qT]$, continuous on $[0, qT]$, such that it verifies the ODE on each open interval $]jT, (j+1)T[$ for $j < q$, and the initial condition $\mathbf{s}(0) = \mathbf{s}_0$.

Example 2: For the ACAS Xu, the temporal evolution of $\mathbf{s}(t)$ can be modelled by the ODE $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ given in equation (1). This ODE is based on a 2D *non-linear* kinematic model where the intruder is assumed to keep constant heading and velocity. This corresponds to a degraded mode where the intruder does not perform any collision avoidance maneuver and continues its uniform rectilinear displacement. For simplicity, the velocity of ownship is also considered constant. It is worth noting that f is continuous in t and u, as well as uniformly Lipschitz continuous in $\mathbf{s}$. Indeed, its



$$\begin{cases} x'(t) = v_{\text{own}}(t)\cdot\sin(\psi_{\text{own}}(t)) - v_{\text{int}}(t)\cdot\sin(\psi_{\text{int}}(t)) \\ y'(t) = v_{\text{int}}(t)\cdot\cos(\psi_{\text{int}}(t)) - v_{\text{own}}(t)\cdot\cos(\psi_{\text{own}}(t)) \\ \psi'_{\text{own}}(t) = u(t) \\ \psi'_{\text{int}}(t) = 0 \\ v'_{\text{own}}(t) = 0 \\ v'_{\text{int}}(t) = 0 \end{cases} \tag{1}$$

Fig. 3. The 2D kinematic model of the ACAS Xu plant $\mathcal{P}$.

derivative *w.r.t.* $\mathbf{s}$ is bounded on $\mathbb{R}\times\mathbb{R}^l\times\mathbb{R}^d$ since both $v_{\text{own}}(t)$ and $v_{\text{int}}(t)$ are constants.

C. Neural network based controller

The controller $\mathcal{N}$ is a classifier based on multiple ReLU networks. More precisely, it involves a collection of *ReLU neural networks* $\mathbf{N} = \{N^{(1)}, \ldots, N^{(D)}\}$ of which only one is executed at each control step. The network $N_j \in \mathbf{N}$ to be executed at step j is selected based on the command $\mathbf{u}_j$ produced at previous step *i.e.*, $N_j = \lambda(\mathbf{u}_j)$ wherein $\lambda : \mathbf{U} \to \mathbf{N}$ maps every command in $\mathbf{U}$ to a network in $\mathbf{N}$. The previous command can thus be seen as the internal state of the controller. Additionally, all the neural networks in $\mathbf{N}$ are assumed to have been trained already, meaning that they remain unchanged for the run-time of the controller.

Definition 2: A *ReLU feedforward deep neural network* is a tuple $N = (L, \{k_l\}_{1\le l\le L}, \mathbf{W}, \mathbf{B})$. It consists of a *directed acyclic weighted graph* where the nodes are arranged in L layers, comprising $k_1, \ldots, k_L$ nodes respectively. The first layer is called the *input layer*, the last layer is called the *output layer*, and the layers in between are called the *hidden layers*. Except the input layer, each layer has its nodes connected to the nodes in the preceding layer. More precisely, let $n_{l,i}$ be the i^{th} node in the l^{th} layer. If $l > 1$, there exists an edge from $n_{l-1,j}$ to $n_{l,i}$ for each $i \in [\![1, k_l]\!]$ and $j \in [\![1, k_{l-1}]\!]$. Moreover, the edge from $n_{l-1,j}$ to $n_{l,i}$ is assigned a *weight* $w^j_{l,i} \in \mathbf{W}$ and each non-input node $n_{l,i}$ is assigned a *bias* $b_{l,i} \in \mathbf{B}$.

This graph actually corresponds to a function $F : \mathbb{R}^{k_1} \to \mathbb{R}^{k_L}$. Indeed, each node $n_{l,i}$ represents a function $F_{l,i}$ the definition of which depends on the layer l. For the nodes in the input layer, this function is the identity function *i.e.*, $F_{1,i} \triangleq \text{id}_{\mathbb{R}},\ \forall i \in [\![1, k_1]\!]$. For the nodes in the hidden layer l, with $1 < l < L$, the associated function maps a vector in $\mathbb{R}^{k_{l-1}}$ to an element in $\mathbb{R}$. It is the composition of a *non-linear ReLU unit* $\sigma : x \mapsto \max(0, x)$ and an *affine transformation* *i.e.*, $F_{l,i} : \mathbf{z} \mapsto \sigma\left(\sum_{j=1}^{k_{l-1}} w^j_{l,i}\cdot\mathbf{z}_j + b_{l,i}\right),\ \forall i \in [\![1, k_l]\!]$. Finally, the function represented by the nodes in the output layer is an affine transformation of a vector in $\mathbb{R}^{k_{L-1}}$ *i.e.*, $F_{L,i} : \mathbf{z} \mapsto \sum_{j=1}^{k_{L-1}} w^j_{L,i}\cdot\mathbf{z}_j + b_{L,i},\ \forall i \in [\![1, k_L]\!]$. Overall, the function computed by the l^{th} layer of the network is the vector function $F_l : \mathbf{z} \mapsto (F_{l,1}(\mathbf{z}) \ \ldots \ F_{l,k_l}(\mathbf{z}))^T$ and the function F computed by the network is the composition function $F \triangleq F_L \circ \ldots \circ F_1$. In particular, F is a deterministic function.

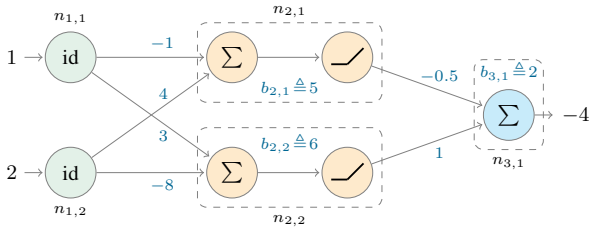


Fig. 4. A (tiny) example ReLU network $N = (3, \{2, 2, 1\}, \mathbf{W}, \mathbf{B})$.

In the example of Fig. 4, the network yields $F_1((1\ 2)) = (1\ \ 2)$ then $F_2((1\ 2)) = (\sigma(-1\times1+4\times2+5)\ \sigma(3\times1-8\times2+6)) = (12\ \ 0)$ and finally $F_3((12\ 0)) = (-0.5\times12+1\times0+2) = -4$.

The j^{th} execution of the controller consists of: (i) a *pre-processing* which selects the network $N_j = \lambda(\mathbf{u}_j)$ to be executed and calculates the input $\mathbf{x}_j \in \mathbb{R}^m$ of the network N_j *i.e.*, $\mathbf{x}_j = \text{Pre}(\mathbf{s}_j)$ wherein $\text{Pre} : \mathbb{R}^l \to \mathbb{R}^m$ (*e.g.*, calculation of a distance from two positions, normalization) (ii) the *neural network execution*, which yields the output vector $\mathbf{y}_j \in \mathbb{R}^p$ such that $\mathbf{y}_j = F_j(\mathbf{x}_j)$ where $F_j : \mathbb{R}^m \to \mathbb{R}^p$ is the function computed by the network N_j, and (iii) a *post-processing* which determines the command $\mathbf{u}_{j+1}$ given the neural network output $\mathbf{y}_j$ *i.e.*, $\mathbf{u}_{j+1} = \text{Post}(\mathbf{y}_j)$ where $\text{Post} : \mathbb{R}^p \to \mathbf{U}$. Typically, each component $(\mathbf{y}_j)_i \in \mathbb{R}$ of the output $\mathbf{y}_j$ could correspond to a command $\mathbf{u}^{(i)} \in \mathbf{U}$, and the post-processing be $\mathbf{u}_{j+1} = \mathbf{u}^{(k)}$ *s.t.* $k = \text{argmin}_i\left((\mathbf{y}_j)_i\right)$. Both the pre and post processing are assumed to be deterministic functions, so that the whole controller is also a deterministic function.

Example 3: To decide on the maneuver to perform, the ACAS Xu controller uses a collection of 5 ReLU networks $\mathbf{N} = \{N^{(1)}, \ldots, N^{(5)}\}$. These networks all have 6 hidden layers of 50 nodes each. They were each trained with supervised learning to approximate a table of the original ACAS Xu, corresponding to one of the 5 possible previous advisories and $t_{\text{sep}} = 0$ (the 40 remaining networks are not considered as they correspond to $t_{\text{sep}} \neq 0$). The pre-processing selects the network to be executed, transforms the sampled state $\mathbf{s}_j$ into the input $\mathbf{x}_j$ by replacing the cartesian coordinates x, y into the cylindrical coordinates ρ, θ (see Fig. 1), and normalizes the resulting vector. The selected neural network outputs 5 scores. Finally, the post-processing chooses the maneuver with the minimal score. A model of the ACAS Xu controller is given

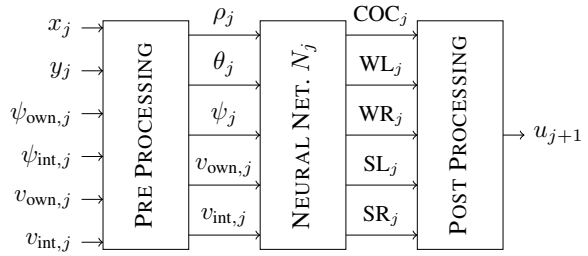


Fig. 5. Model of the neural network based ACAS Xu controller.

in Fig. 5.

D. Safety verification problem

Given a closed-loop system $\mathcal{C}$ and its evolution over a finite time horizon, the safety verification purpose is to prove that no unsafe state can be reached. For that, we consider a set of *erroneous* states $\mathbf{E} \subset \mathbb{R}^l \times \mathbf{U}$ such that a state $\phi(t) \in \mathbf{E}$ causes a potentially catastrophic failure of $\mathcal{C}$. It is thus expected that $\mathcal{C}$ does not reach a state in $\mathbf{E}$. We also assume that $\mathcal{C}$ terminates when its state $\phi(t)$ belongs to a set $\mathbf{T} \subset \mathbb{R}^l \times \mathbf{U}$, with $\mathbf{T} \cap \mathbf{E} = \emptyset$ to ensure a safe behaviour. Here, $\mathbf{T}$ can be seen as a set of *target* states, corresponding to $\mathcal{C}$ having successfully achieved its mission. It is thus expected that $\mathcal{C}$ terminates in a *finite* amount of time, whatever the initial state. We denote by $\tau \in \mathbb{R}$ the expected (or estimated) upper bound on this amount of time, independently of the initial state. Additionally, we set by definition $\phi(t) = \perp$ after the termination of the closed-loop system $\mathcal{C}$ *i.e.*, if $t_{\text{end}} \leq \tau$ satisfies $\forall t < t_{\text{end}},\ \phi(t) \notin \mathbf{T}$ and $\phi(t_{\text{end}}) \in \mathbf{T}$ then $\phi(t) = \perp\ \forall t \in]t_{\text{end}}, \tau]$. In other words, the bottom element symbolically represents the "terminated" state of $\mathcal{C}$.

Example 4: Back to the ACAS Xu system, we consider a set $\mathbf{E}$ of erroneous states representing a collision between the two aircraft. Such a collision happens when the intruder enters the collision circle around ownship, with a radius of 500 ft [MJ16], hence $\mathbf{E} = \{\phi(t) = (\mathbf{s}(t), u(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \sqrt{x(t)^2 + y(t)^2} < 500.0 \text{ ft}\}$. Finally, the closed-loop system terminates when the intruder leaves the circle $\mathcal{R}$ *i.e.* the ownship does not see it anymore: $\mathbf{T} = \{\phi(t) = (\mathbf{s}(t), u(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \sqrt{x(t)^2 + y(t)^2} > r\}$. In particular, $\mathcal{C}$ terminates in a finite amount of time and the value of τ can be estimated from v_{own}, v_{int} and r.

*a) **Reachability definition.**:* As the combination of a deterministic plant $\mathcal{P}$ and a deterministic controller $\mathcal{N}$ (see sections III-B and III-C), the closed-loop system $\mathcal{C}$ has a deterministic behaviour. More precisely, for a given initial state $\phi_0 \in \mathbf{I}$, there exists a unique function $\phi_{\phi_0} : [0, \tau] \to \mathbb{R}^l \times \mathbf{U} \cup \{\perp\}$ such that $\phi_{\phi_0}(t)$ is the state of $\mathcal{C}$ at instant $t \leq \tau$. This hypothesis is important for properly defining the verification problem, as well as demonstrating the soundness of our procedure.

Definition 3: The reachable states of the closed-loop system $\mathcal{C}$ at a given instant $t \leq \tau$ is the set $\mathbf{R}_t = \{\phi \in \mathbb{R}^l \times \mathbf{U} \cup \{\perp\} \mid \exists \phi_0 \in \mathbf{I},\ \phi = \phi_{\phi_0}(t)\}$.

Definition 4: The reachable states of the closed-loop system $\mathcal{C}$ for the time interval $[t_1, t_2] \subset [0, \tau]$ (resp. $[t_1, t_2[\subset [0, \tau]$) is the set $\mathbf{R}_{[t_1,t_2]} = \{\phi \in \mathbf{R}_t \mid t \in [t_1, t_2]\}$ (resp. $\mathbf{R}_{[t_1,t_2[} = \{\phi \in \mathbf{R}_t \mid t \in [t_1, t_2[\}$).

*b) **Problem definition.**:* We want to *decide* if, whatever the initial state ϕ_0 in $\mathbf{I}$, the closed-loop system $\mathcal{C}$ remains safe *w.r.t.* the set of erroneous states $\mathbf{E}$ over the time horizon τ. In other words, we want to decide if the reachable states of $\mathcal{C}$ in $[0, \tau]$ remain outside $\mathbf{E}$.

Definition 5: The *safety verification problem* $\mathcal{V}$ consists in deciding if:

$$\mathbf{R}_{[0,\tau]} \cap \mathbf{E} = \emptyset \tag{2}$$

Reasoning about the problem $\mathcal{V}$ is a difficult task. Indeed, whatever the nature of the controller (based on ReLU networks or not), the problem $\mathcal{V}$ is undecidable when the plant $\mathcal{P}$ has a non-linear dynamics [ACH+95], [Hai08] (*e.g.*, ACAS Xu). Furthermore, the neural networks add to the complexity of the verification problem. Indeed, due to the *non-linear* ReLU units and the *many dependencies* induced by the affine transformations, the function computed by a ReLU network is non-monotonic, non convex and highly non-linear. As a result, its behaviour is very difficult to analyze for a *continuum* of inputs, which is the case in problem $\mathcal{V}$ as the initial set $\mathbf{I}$ is infinite. Actually, it has been shown that verifying pre/post-conditions on a ReLU network is a NP-hard problem [KBD+17]. Finally, the controller has a non-trivial logic, switching between the networks and involving pre and post-processing stages, which increases the dependencies from one control step to another.

As the problem $\mathcal{V}$ is undecidable, we aim at constructing a *sound* approximation of the reachable states of $\mathcal{C}$. More precisely, we aim at computing a *bounded* set $\widetilde{\mathbf{R}}_{[0,\tau]}$ satisfying $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$. Indeed, provided we are able to compute such a set and if it verifies $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$, then (2) is proved to hold. Consequently, we consider the problem $\widetilde{\mathcal{V}}$ defined as follows:

Definition 6: The *safety verification problem* $\widetilde{\mathcal{V}}$ consists in finding a set $\widetilde{\mathbf{R}}_{[0,\tau]}$ satisfying $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$ and $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$.

IV. Reachability-based approach

This section describes a *tight* over-approximation approach to compute $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$ in order to find a solution to problem $\widetilde{\mathcal{V}}$.

A. Symbolic state and symbolic set

The set $\widetilde{\mathbf{R}}_{[0,\tau]}$ that we aim at constructing is *infinite*. To allow reasoning about this type of set, we introduce the notions of *symbolic state* and *symbolic set*.

Definition 7: A *symbolic state* is a 2-tuple $([\mathbf{s}], \mathbf{u})$ wherein $[\mathbf{s}] \subset \mathbb{R}^l$ is a l-dimensional box *i.e.*, the cartesian product of l intervals, and $\mathbf{u} \in \mathbf{U}$. It symbolically represents the set $\{\phi(t) = (\mathbf{s}(t), \mathbf{u}(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \mathbf{s}(t) \in [\mathbf{s}]\ \wedge\ \mathbf{u}(t) = \mathbf{u}\}$.

Example 5: For the ACAS Xu, the symbolic state $([\mathbf{s}], u)$ with $[\mathbf{s}] = [-20ft, 0ft] \times [8000ft, 8500ft] \times [0, 0] \times [3.10, 3.14] \times [700ft/s, 700ft/s] \times [600ft/s, 600ft/s]$ and

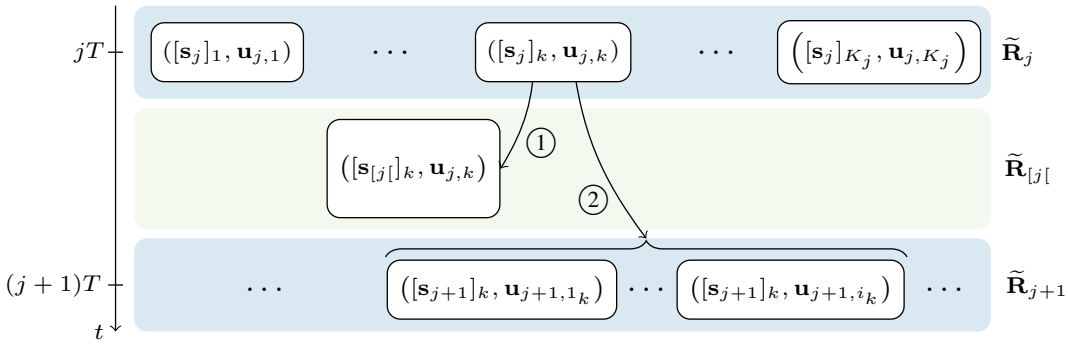


Fig. 6. The reachability procedure at control step j, where ① involves validated simulation and ② involves both validated simulation and abstract interpretation.

$u = 0.0deg/s$ represents a (infinite) set of states where the intruder is ahead of ownship, moving towards the ownship, and the controller advises COC.

Definition 8: A *symbolic set* is a collection of symbolic states defined by $\widetilde{\mathbf{\Phi}} = \{([\mathbf{s}]_k, \mathbf{u}_k)\}_{1 \leq k \leq K}$ wherein $K \in \mathbb{N}$. It corresponds to the union of the sets represented by each $([\mathbf{s}]_k, \mathbf{u}_k)$.

As one can note, a symbolic set can be used to symbolically approximate *any* set of (non-bottom) states of $\mathcal{C}$ (the bottom element is not considered as it does not impact safety). Moreover, our definition yields a rather accurate approximation as it captures the dependency between the state $\mathbf{s}(t)$ of the plant $\mathcal{P}$ and the actuation command $\mathbf{u}(t)$ from the controller. In the following, we extend the set operations and relations to both symbolic states and symbolic sets *e.g.*, $\phi \in \widetilde{\mathbf{\Phi}}$ iff ϕ belongs to the set represented by $\widetilde{\mathbf{\Phi}}$.

B. Over-approximation techniques

We rely on existing over-approximation techniques to construct $\widetilde{\mathbf{R}}_{[0,\tau]}$. More precisely, *validated simulation* soundly approximates the plant dynamics and *abstract interpretation* soundly approximates the controller behaviour.

Validated simulation. Let us consider an ODE $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ wherein $\mathbf{s} : \mathbb{R} \to \mathbb{R}^l$, $\mathbf{u} : \mathbb{R} \to \mathbb{R}^d$ is a given function, continous in t, and $f : \mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d \to \mathbb{R}^l$ is assumed to be continuous in t and $\mathbf{u}$ and uniformly Lipschitz continuous in $\mathbf{s}$. Moreover, let us consider an interval $[t_1, t_2]$ and a l-dimensional box $[\mathbf{s}_{t=t_1}] \subset \mathbb{R}^l$ representing a set of initial values. The goal of validated simulation is to over-approximate the reachable solutions of the ODE satisfying $\mathbf{s}(t = t_1) \in [\mathbf{s}_{t=t_1}]$, over the whole time interval $[t_1, t_2]$. More precisely, it aims at computing the l-box $[\mathbf{s}_{[t_1,t_2]}] \subset \mathbb{R}^l$ approximating the reachable values of $\mathbf{s}(t)$ for $t \in [t_1, t_2]$, and the tighter l-box $[\mathbf{s}_{t=t_2}] \subset [\mathbf{s}_{[t_1,t_2]}]$ approximating the reachable values of $\mathbf{s}(t)$ at $t = t_2$. Consequently, if $\mathbf{s}$ satisfies the ODE and the initial condition $\mathbf{s}(t = t_1) \in [\mathbf{s}_{t=t_1}]$ then $\big(\mathbf{s}(t) \in [\mathbf{s}_{[t_1,t_2]}]\ \forall t \in [t_1, t_2]\big) \wedge (\mathbf{s}(t = t_2) \in [\mathbf{s}_{t=t_2}])$. Usually, validated simulation is based on the 2-step Lohner type algorithm: the enclosure $[\mathbf{s}_{t_1,t_2}]$ is calculated using the Banach fixed point theorem while the enclosure $[\mathbf{s}_{t=t_2}]$ is computed based on a numerical integration method (*e.g.*, Euler, Runge-Kutta) and the associated local truncation error [AdSC16].

Abstract interpretation. Let us consider a function $F : \mathbb{R}^m \to \mathbb{R}^p$ and let $[\mathbf{x}] \subset \mathbb{R}^m$ be a m-dimensional box representing a set of inputs. The goal of abstract interpretation is to soundly approximate the set of the reachable outputs from $[\mathbf{x}]$ *i.e.*, the set $F([\mathbf{x}]) = \{F(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\}$. To this end, abstract interpretation leverages an abstract transformer $F^{\#}$ that soundly approximates the semantics of F. Intuitively, it "propagates" $[\mathbf{x}]$ through the function F. This yields the p-box $[\mathbf{y}] = F^{\#}([\mathbf{x}])$ satisfying $[\mathbf{y}] \supset F(\mathbf{X})$. The abstract transformer $F^{\#}$ can rely on interval arithmetics or affine arithmetics for example [SdF03].

C. Procedure

We consider that the finite window τ comprises q executions of the controller *i.e.*, $\tau = qT$. The overall idea is to *iteratively* build the set $\widetilde{\mathbf{R}}_{[0,\tau]}$, based on the successive executions of the controller. The procedure involves two types of sets:

(a) The symbolic set $\widetilde{\mathbf{R}}_j \supset \mathbf{R}_{jT} \setminus \{\bot\}$ approximates the (non-bottom) reachable states at $t = jT$, with $j \leq q$. The k^{th} symbolic state composing $\widetilde{\mathbf{R}}_j$ is denoted $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$. It represents a set of states $\mathbf{s}(t)$ that are reachable together with the command $\mathbf{u}_{j,k}$ at $t = jT$.

(b) The symbolic set $\widetilde{\mathbf{R}}_{[j[} \supset \mathbf{R}_{[jT,(j+1)T[} \setminus \{\bot\}$ approximates the (non-bottom) reachable states for $t \in [jT, (j+1)T[$, with $j < q$. The k^{th} symbolic state composing $\widetilde{\mathbf{R}}_{[j[}$ is denoted $([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})$. It represents a set of states $\mathbf{s}(t)$ that are reachable together with the command $\mathbf{u}_{j,k}$ for $t \in [jT, (j+1)T[$.

The procedure starts with the symbolic set $\widetilde{\mathbf{R}}_0 \supset \mathbf{R}_0 = \mathbf{I}$ enclosing the possible initial states. Then, for $j \in [\![0, q-1]\!]$, it computes the reachable symbolic states from each symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ composing $\widetilde{\mathbf{R}}_j$ (see Fig. 6). More precisely, for each $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j$, it computes:

(1) The symbolic state $([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})$ approximating the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ over $[jT, (j+1)T[$, where $[\mathbf{s}_{[j[}]_k$ is calculated using validated simulation and $\mathbf{u}_{j,k}$ is the constant actuation command over $[jT, (j+1)T[$. To compute $[\mathbf{s}_{[j[}]_k$, we consider the ODE $\mathbf{s}'(t) =$

$f(t, \mathbf{s}(t), \mathbf{u}(t))$ and the time interval $[jT, (j+1)T]$, with $\mathbf{s}(t = jT) \in [\mathbf{s}_j]_k$ and $\mathbf{u}(t) = \mathbf{u}_{j,k}\ \forall t \in [jT, (j+1)T]$. Validated simulation is used to compute the l-box $[\mathbf{s}_{[jT,(j+1)T]}]$ enclosing the reachable values of $\mathbf{s}(t)$ for $t \in [jT, (j+1)T]$. Then we take $[\mathbf{s}_{[j[}]_k = [\mathbf{s}_{[jT,(j+1)T]}]$ which is sound as $[jT, (j+1)T[\subset [jT, (j+1)T]$.

(2) The symbolic states $([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k})$ approximating the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ at $t = (j+1)T$, where $[\mathbf{s}_{j+1}]_k$ is calculated using validated simulation and the reachable commands $\mathbf{u}_{j+1,1_k}, \ldots, \mathbf{u}_{j+1,i_k}$ are calculated using abstract interpretation. To compute $[\mathbf{s}_{j+1}]_k$, we consider the same hypotheses as in (1) except that validated simulation is used to compute the l-box $[\mathbf{s}_{t=(j+1)T}]$ enclosing the reachable values of $\mathbf{s}(t)$ at $t = (j+1)T$. Then we take $[\mathbf{s}_{j+1}]_k = [\mathbf{s}_{t=(j+1)T}]$ which is sound even though the actuation command may have changed at $t = (j+1)T$ (this is due to the continuity of $\mathbf{s}$). Additionally, the selected controller $N_{j,k}$ (obtained from the previous command *i.e.*, $N_{j,k} = \lambda(\mathbf{u}_{j,k})$) is approximated thanks to abstract interpretation: (i) the m-box $[\mathbf{x}_j]_k = \text{Pre}^{\#}([\mathbf{s}_j]_k)$ approximates the reachable inputs, (ii) the p-box $[\mathbf{y}_j]_k = F^{\#}_{j,k}([\mathbf{x}_j]_k)$ approximates the reachable outputs and (iii) the finite set $\{\mathbf{u}_{j+1,1_k}, \ldots, \mathbf{u}_{j+1,i_k}\} = \text{Post}^{\#}([\mathbf{y}_j])$ approximates the reachable commands at $t = (j+1)T$.

By definition, the stages (1) and (2) yield the symbolic sets $\widetilde{\mathbf{R}}_{[j[}$ and $\widetilde{\mathbf{R}}_{j+1}$, the latter being used in the next iteration. Finally, the q^{th} iteration yields $\widetilde{\mathbf{R}}_{[0,\tau]} = \cup_{0 \le j < q} \widetilde{\mathbf{R}}_{[j[} \cup \widetilde{\mathbf{R}}_q$.

Actually, to take account of a potential termination of $\mathcal{C}$, we consider a slight variant of the above procedure. Indeed, if a symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ composing $\widetilde{\mathbf{R}}_j$ satisfies $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \subset \mathbf{T}$, then this symbolic state is not further propagated *i.e.*, the reachable symbolic states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ are not computed. Consequently, if there exists $j^{\text{end}} \le q$ such that there is no more symbolic state to be propagated from $\widetilde{\mathbf{R}}_{j_{\text{end}}}$, then we take $\widetilde{\mathbf{R}}_{[0,\tau]} = \cup_{0 \le j < j^{\text{end}}} \widetilde{\mathbf{R}}_{[j[} \cup \widetilde{\mathbf{R}}_{j^{\text{end}}}$. Moreover, if $\widetilde{\mathbf{R}}_{[0,\tau]}$ satisfies $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$, then the closed-loop $\mathcal{C}$ is proved to be safe *until it terminates*.

Theorem 1: The procedure yields a sound approximation of the non-bottom reachable states *i.e.*, $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]} \setminus \{\bot\}$.

D. Implementation details

The procedure has been implemented as a Python program that interfaces with existing tools. The validated simulation of the plant dynamics is based on DynIBEX [AdSC16]. The abstract transformers for the pre- and post-processing are based on interval arithmetics, which is easy to implement, computationally efficient and accurate for simple functions. The abstract transformer for the neural networks relies on dedicated tools, which have been slightly adapted to fit our approach: either ReluVal [WPW+18] or DeepPoly [SGPV19].

V. Experiments

This section presents our results for the verification of the ACAS Xu system described in section III. The experiment was conducted using the DeepPoly abstract transformer and with $r = 8000$ ft for the range of the ownship sensors and $v_{\text{own},0} = 700$ ft/s, $v_{\text{int},0} = 600$ ft/s for the initial velocities of the ownship and the intruder respectively. The experiment was run on CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM.

*a) **Partitioning**:* For verifying the ACAS Xu, we used an empirical partitioning of the possible initial states. More precisely, the circle $\mathcal{R}$ was partitioned into 629 arcs of length 80 ft each. Additionally, for each arc, the possible initial headings $\psi_{\text{int},0}$ of the intruder were partitioned into 316 subsets of size 0.01 rad each. With the initial heading of ownship $\psi_{\text{own},0}$ and the initial velocities $v_{\text{own},0}$, $v_{\text{int},0}$ being fixed, we obtained a partition of size $K_0 = 198,764$ of the possible initial states $\mathbf{s}_0$ of the plant $\mathcal{P}$. Then, each element of this partition was over-approximated by a 5-dimensional box $[\mathbf{s}_0]_k \subset \mathbb{R}^5$, with $1 \le k \le K_0$. Finally, we took as input for the procedure the symbolic set $\widetilde{\mathbf{R}}_0 = \{([\mathbf{s}_0]_k, 0.0 \text{ deg/s})\}_{1 \le k \le K_0}$. An initial symbolic state for which the system could not be proved safe was split into smaller initial symbolic states: $[\mathbf{s}_0]_k$ was bisected along the dimensions corresponding to x_0, y_0 and $\psi_{\text{int},0}$, yielding 2^3 new initial symbolic states. This split refinement process was repeated iteratively until the system could be proved safe, with a maximum depth of 1 split.

*b) **Results**:* We recorded the *coverage* c representing the percentage of the possible initial states for which the ACAS Xu was proved safe until it terminates *i.e.*, $c = 100/K_0 \cdot \sum_{d=0}^{1} n_d/(2^3)^d$ wherein n_d is the number of initial symbolic states resulting from d split refinements for which the ACAS Xu was proved safe. The experiment took about 28 hours and yielded a coverage $c = 98.8\%$, meaning that the ACAS Xu was proved safe for 98.8% of the possible initial states and for the remaining states, we could not prove it safe. This is still a valuable information as one can design a real-time monitoring mechanism that switches to a more robust controller when the ACAS Xu encounters an initial state for which it was not proved safe. Having such an architecture would allow to benefit from the expected performance of the neural networks while still remaining safe. The large verification time is partly due to the difficulty of the verification problem since the set of the initial states is quite large.

VI. Conclusion and future work

This paper presented a technique to verify the safety requirements of complex neural network controlled systems such as the ACAS Xu. We evaluated the applicability of our approach by providing the first sound guarantees of safety of the overall neural network based ACAS Xu.

For future work, we aim at reducing the verification time by using a more efficient partitioning strategy of the initial states and by optimizing the reachability procedure. Another direction is to consider a more complex ACAS Xu system,

where both the ownship and the intruder are equipped with the controller, or with more than 2 UAVs. A third direction is to provide a thorough comparison with state-of-the-art tools based on additional use cases such as VCAS (Vertical Collision Avoidance System).

REFERENCES

[ABKL20] Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. Verifying strategic abilities of neural-symbolic multi-agent systems. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR'20)*, pages 22–32, 2020.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 138:3–34, 1995.

[AdSC16] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing electronic edition*, 22, July 2016.

[ALFS11] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, pages 254–257, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Alt15] Matthias Althoff. An introduction to cora 2015. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 120–151, 2015.

[BFG+19] Sergiy Bogomolov, Goran Frehse, Amit Gurung, Dongxu Li, Georg Martius, and Rajarshi Ray. Falsification of hybrid systems using symbolic reachability and trajectory splicing. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 1–10, 2019.

[BTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[CAGP20] Arthur Clavière, Eric Asselin, Christophe Garion, and Claire Pagetti. Safety verification of neural network controlled systems. *CoRR*, abs/2011.05174, 2020.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2013.

[CSKX15] Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *CoRR*, abs/1505.00256, 2015.

[DCS19] 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 '19, page 157–168, New York, NY, USA, 2019.

[Ehl17] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320, 2017.

[EUR10] EUROCAE/SAE. Aerospace Recommended Practices ARP4754a - development of civil aircraft and systems, 2010. SAE.

[EUR11] EUROCAE/RTCA, Inc. DO-178 ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.

[EUR20] EUROCAE/RTCA. EUROCAE WG 75.1 / RTCA SC-147 - Minimum Operational Performance Standards For Airborne Collision Avoidance, 2020.

[GMDC+18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[Hai08] Emmanuel Hainry. Reachability in linear dynamical systems. In *Logic and Theory of Algorithms*, pages 241–250, 2008.

[HFL+19] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN: Reachability Analysis of Neural-Network Controlled Systems, 2019.

[HKR+20] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability, 2020.

[IWA+18] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. *CoRR*, abs/1811.01828, 2018.

[JK19] Kyle D. Julian and Mykel J. Kochenderfer. Guaranteeing safety for neural network-based aircraft collision avoidance systems. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019.

[JKO18] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *CoRR*, abs/1810.04240, 2018.

[KBD+17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

[LMT+19] 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 *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*, pages 103–119, 2019.

[MJ16] Guido Manfredi and Yannick Jestin. An introduction to acas xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9, 2016.

[PCS+17] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *CoRR*, abs/1709.07174, 2017.

[SdF03] J. Stolfi and L.H. de Figueiredo. An introduction to affine arithmetic. *TEMA (São Carlos)*, 4(3):297–312, 2003.

[SGM+18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 10802–10813. Curran Associates, Inc., 2018.

[SGPV19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), 2019.

[TYL+20] 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 *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, pages 3–17, 2020.

[WPW+18] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security 2018*, pages 1599–1614, 2018.