Neural Network Compression of ACAS Xu Early Prototype is Unsafe: Closed-Loop Verification through Quantized State Backreachability

Stanley Bak [1] **Hoang-Dung Tran** [2]

Abstract

ACAS Xu is an air-to-air collision avoidance system designed for unmanned aircraft that issues horizontal turn advisories to avoid an intruder aircraft. Analysis of this system has spurred a significant body of research in the formal methods community on neural network verification. While many powerful methods have been developed, most work focuses on open-loop properties of the networks, rather than the main point of the system—collision avoidance—which requires closed-loop analysis. In this work, we develop a technique to verify a closed-loop approximation of the system using *state quantization* and *backreachability*. We use favorable assumptions for the analysis—perfect sensor information, instant following of advisories, ideal aircraft maneuvers and an intruder that only flies straight. When the method fails to prove the system is safe, we refine the quantization parameters until generating counterexamples where the original (non-quantized) system also has collisions.

1. Introduction

The Airborne Collision Avoidance System X (ACAS X) is a mid-air collision avoidance system under development (Olson, 2015), with the ACAS Xu variant focused on collision avoidance for unmanned aircraft (Katz et al., 2017). Originally designed offline using dynamic programming and Markov decision processes (MDPs) (Kochenderfer & Chryssanthacopoulos, 2011), the large rule table was compressed by a factor of 1000 using a set of neural networks (Julian et al., 2016). As collision avoidance is safety-critical, analysis of the neural networks has spurred a significant body of research on neural network verification. Most existing work, however, focuses on *open-loop* verification, such as property ϕ_3 from the original work (Katz et al., 2017), which states, "if the intruder is directly ahead and is moving towards the ownship, [a turn will be commanded]." Open-loop properties can be expressed in terms of constraints over the inputs and outputs of a single execution of the neural network. However, satisfying open-loop properties does not prove the system is safe, as this requires reasoning with the physical system dynamics—how the aircraft responds to turn commands. Also, the system is running continuously and may change advisories at a future time, complicating safety analysis. Verification of closed-loop safety of provided collision avoidance system under all designed operating conditions is thus a sort of grand challenge. While verification of neural networks is continuously improving, an intriguing alternate approach has recently been proposed based on input quantization (Jia & Rinard, 2021). Rather than verifying the neural network directly, which requires reasoning about the semantics at each layer, the system's execution semantics are changed to round the inputs to a discrete set of possible values before running the network.

In this work, we propose an approach to formally verify quantized closed-loop NNCS. Although the technique is general, we focus primarily on proving safety for quantized version of the well-studied aircraft collision avoidance neural network benchmark. Two key ideas are needed to make this work: (1) we perform *state quantization* rather than input quantization and (2) we use *backreachability* from the unsafe states to reduce the number of partitions. We prove the approach is sound and complete, in the sense that by continuing to refine quantization parameters, either the quantized system will eventually be proven safe or an unsafe counterexample will be found in the original system. When the method fails to prove safety of quantized closed-loop system, we refine the quantization values until discovering cases where the original (unquantized) version of the system fails. We also show that with stricter assumptions on the ownship aircraft's velocity, the quantized system can guarantee safety.

[*]Equal contribution [1]Department of Computer Science, Stony Brook University, New York, USA [2]School of Computing, University of Nebraska-Lincoln, Lincoln, USA. Correspondence to: Stanley Bak <stanley.bak@stonybrook.edu>, Hoang-Dung Tran <dtran30@unl.edu>.

1[st] Workshop on Formal Verification of Machine Learning, Baltimore, Maryland, USA. Colocated with ICML 2022.

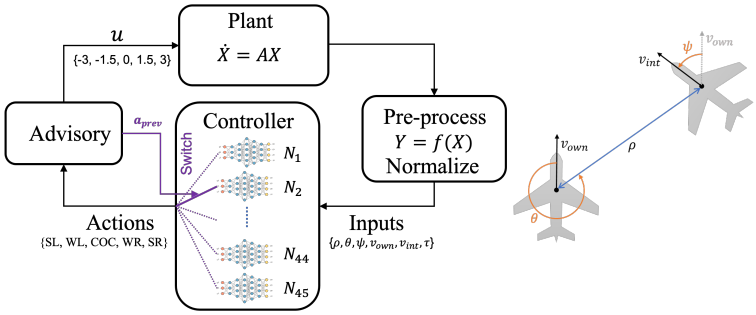


Figure 1. The closed-loop air-to-air collision avoidance system.

2. Background and Problem Formulation

We next review key aspects of the system design, proof assumptions, and provide background on $\mathcal{AH}$-Polytopes before formulating the safety verification problem.

2.1. Collision Avoidance System Design

We are interested in safety verification and falsification of the *closed-loop* air-to-air collision avoidance system (Kochenderfer & Chryssanthacopoulos, 2011; Katz et al., 2017) depicted in Figure 1.

A detailed description of the inputs and actions in the system is shown in Table 1. The system receives 7 inputs about the state of an ownship and a nearby intruder aircraft, $\mathcal{I} = \{\rho, \theta, \psi, v_{own}, v_{int}, \tau, a_{prev}\}$, and produces one of five possible advisories for the ownship, $\mathcal{A} = \{\text{COC}, \text{WL}, \text{WR}, \text{SL}, \text{SR}\}$. The turn advisories in the system are generated by 45 deep ReLU neural networks with 6 layers and 50 neurons per layer for each network. Control switches between different neural networks $N_{a_{prev},\tau}$ based on the previous advisory a_{prev} (total of 5 choices) and the time until loss of vertical separation $\tau = \{0, 1, 5, 10, 20, 50, 60, 80, 100\}$ (total of 9 choices). For example, the network $N_{5,3}$ will be invoked if the previous advisory is $a_{prev} = \text{SR}$ and $\tau = 5$. If the ownship and the intruder are at the same altitude, then $\tau = 0$ and only five neural network controllers need to be used, $N_{1,1}, N_{2,1}, N_{3,1}, N_{4,1}$, and $N_{5,1}$.

2.2. Assumptions and Plant Model

Before we describe the plant model used in analysis, we first state our system assumptions: (i) the intruder flies in straight-line trajectories with constant speed, (ii) the ownship flies with constant speed and its heading is adjusted every second (the NNCS control period), (iii) the actions correspond to heading changes in the intruder of 1.5 deg/sec for weak turn commands, 3.0 deg/sec for strong turns and 0.0 deg/sec for clear-of-conflict commands (Julian et al., 2016), (iv) there is no sensor noise and (v) advisories are followed exactly and immediately. To model the state of the system with these assumptions, we use Cartesian coordinates. The values $x_{own}, y_{own}, x_{int}, y_{int}$ refer to the x and y positions of the ownship and the intruder; $v_{own} = \sqrt{(v^x_{own})^2 + (v^y_{own})^2}$ and $v_{int} = \sqrt{(v^x_{int})^2 + (v^y_{int})^2}$ are the speed of the ownship and the intruder; θ_{own} and θ_{int} are the heading of the ownship and the intruder w.r.t the x axis. The system performs idealized turn maneuvers modeled with Dubins aircraft dynamics:

$$\begin{aligned} \dot{x}_{own} &= v^x_{own} = v_{own} cos(\theta_{own}) \\ \dot{y}_{own} &= v^y_{own} = v_{own} sin(\theta_{own}) \\ \dot{x}_{int} &= v^x_{int} = v_{int} cos(\theta_{int}) \\ \dot{y}_{int} &= v^y_{int} = v_{int} sin(\theta_{int}) \end{aligned} \tag{1}$$

Equation 1 does not show clearly how the aircraft can be controlled by changing their heading. Taking derivatives of the Equation 1 one more time and noticing that $\dot{\theta}_{own}$ is a constant between advisories, $\dot{\theta}_{own} = (\pi/180)u = c(rad/s)$, and then taking $\dot{\theta}_{int} = 0$, we obtain the following 8-d linear system dynamics:

$$\begin{bmatrix} \dot{x}_{own} \\ \dot{y}_{own} \\ \dot{v}^x_{own} \\ \dot{v}^y_{own} \\ \dot{x}_{int} \\ \dot{y}_{int} \\ \dot{v}^x_{int} \\ \dot{v}^y_{int} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -c & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{own} \\ y_{own} \\ v^x_{own} \\ v^y_{own} \\ x_{int} \\ y_{int} \\ v^x_{int} \\ v^y_{int} \end{bmatrix} \tag{2}$$

The linear model described in Equation 2 is valid for only one control step, with a fixed control signal u, which may be either $-3, -1.5, 0, 1.5$ or 3 deg/s depending on the specific command. Therefore, this model can be considered as a piece-wise linear model of the system. From the plant state variables, we can obtain the inputs for the neural network controller which are expected to in radial coordinates as

Input	Units	Description	Action	Description
ρ	ft	distance between ownship and intruder	SL	strong left turn at 3.0 deg/s
θ	rad	angle to intruder w.r.t ownship heading	WL	weak left at turn 1.5 deg/s
ψ	rad	heading of intruder w.r.t ownship	COC	clear of conflict (do nothing)
v_{own}	ft/s	velocity of ownship	WR	weak right turn at 1.5 deg/s
v_{int}	ft/s	velocity of intruder	SR	strong right turn at 3.0 deg/s
τ	s	time until loss of vertical separation		
a_{prev}		previous advisory		

Table 1. Input variables used to produce a turn advisory.

follows.

$$\theta_{own} = arctan(\frac{v^y_{own}}{v^x_{own}}), \quad \theta_{int} = arctan(\frac{v^y_{int}}{v^x_{int}}),$$
$$\rho = \sqrt{(x_{int} - x_{own})^2 + (y_{int} - y_{own})^2},$$
$$\theta = arctan(\frac{y_{int} - y_{own}}{x_{int} - x_{own}}) - \theta_{own}, \quad \psi = \theta_{int} - \theta_{own}.$$

2.3. Reachability with $\mathcal{AH}$-Polytopes

An $\mathcal{AH}$-polytope is a set representation that informally is an affine transformation of a half-space polytope, where the affine transformation and polytope terms are explicitly kept separate. Although the name is fairly recent (Sadraddini & Tedrake, 2019), this set representation has often been used in reachability analysis for linear systems (Bak et al., 2019; Bak & Duggirala, 2017) and neural networks (Tran et al., 2019b; Bak et al., 2020), where it is also called a linear star set (Duggirala & Viswanathan, 2016), constrained zonotope (Scott et al., 2016), affine form (Han & Krogh, 2006), or symbolic orthogonal projection (Hagemann, 2014).

Importantly for this work, discrete-time reachability of systems with linear dynamics, $\dot{x} = Ax$, can be expressed exactly using this set representation, as it amounts to a linear transformation of the entire set by the matrix exponential e^{At}, where t is the time step. Further, operations like intersections can be performed exactly on $\mathcal{AH}$-polytopes, as well as linear optimization over the sets. A formal definition and operation list is provided in Appendix.

2.4. Safety Problem Formulation

Verifying the safety of the closed-loop system means proving the absence of *unsafe paths* under all operating conditions. For simplified presentation, we consider a discrete-time version of the problem, where we only check for collisions once a second when the system is activated. Our analysis could be extended to continuous time through *conservative time-discretization* approaches from hybrid systems reachability analysis (Forets & Schilling, 2021), which essentially bloat the initial set and then perform discrete-time analysis.

Definition 2.1 (Path)**.** A path is written as $s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_{n-1}} s_n$, where successive values of s_i and s_{i+1} correspond to the state of the system one second apart according to the plant dynamics in Equation 2. The command α_i is the system output from state s_i using $\alpha_{\text{prev}} = \alpha_{i-1}$, with s_1 using the COC network. Paths can either be in-plane, where $\dot{\tau} = 0$ and $\tau = 0$ in all states and so the $N_{1,*}$ networks get used to generate all commands, or out-of-plane, where $\dot{\tau} = -1$. In the out-of-plane case, each state in the path should decrease τ by one second.

An unsafe path has s_1 as an initial state and s_n as an unsafe state.

Definition 2.2 (Initial State)**.** An initial state of the state of the system is one where the aircraft are outside of the system's operating range ($\rho > 60760$ ft).

Definition 2.3 (Unsafe State)**.** Unsafe states are defined to be any states in the near mid-air collision (NMAC) cylinder (Marston & Baca, 2015), where the horizontal separation ρ is less than 500 ft and the time to loss of vertical separation τ is zero seconds.

The operating conditions where the system should ensure safety are extracted based on the training ranges used for the original neural networks (Kochenderfer & Chryssanthacopoulos, 2011; Katz et al., 2017). The system should be active when the distance between aircraft $\rho \in [0, 60760]$ ft, otherwise clear-of-conflict is commanded. The valid values for the ownship velocity are $v_{own} \in [100, 1200]$ ft/sec, valid values for intruder velocity are $v_{int} \in [0, 1200]$ ft/sec, and the angular inputs θ and ψ are both between $-\pi$ and π.

3. Quantized State Backreachability

Our verification strategy is to compute the backwards reachable set of states from all possible unsafe states, trying to a find a path that begins with an initial state. We first partition the unsafe states along state quantization boundaries.

3.1. Partitioning the Unsafe States

Since the system advisories are only based on relative positions and headings, we eliminate symmetry by assuming that at the time of the collision the intruder is flying due east and at the origin. We then consider all possible posi-

tions of the ownship to account for all possible unsafe states. Three quantization parameters are used in the analysis: q_{pos} to quantize positions, q_{vel} to quantize velocities, and q_θ to quantize the heading angle. Based on these parameters, we partition the unsafe states into 8-d $\mathcal{AH}$-polytopes covering the entire set of possible unsafe states. The eight dimensions correspond to the system states in the linear dynamics in Equation 2, including positions x, y, and velocities v^x, v^y for both the ownship and intruder. Associated with each partition, we also enumerate the five possible previous commands α_{prev} and two possibilities for whether there is a relative vertical velocity—whether the time to loss of vertical separation is fixed at 0 or decreasing, $\dot{\tau} \in \{0, -1\}$.

To create partitions, the x_{own} and y_{own} values are divided into a grid based on q_{pos}. The intruder position (x_{int}, y_{int}) is set to $(0, 0)$. The intruder and ownship velocities are partitioned based on q_{vel}, which gets reflected in the x and y velocity state variables for the two aircraft. The intruder is moving due east, so $v^y_{int} = 0$ and v^x_{int} is set to the range of intruder velocities corresponding to the current partition. The heading of the angle of the ownship is partitioned based on q_θ, where each partition has a lower and upper bound on the heading $[\theta^{lb}_{own}, \theta^{ub}_{own}]$. From the current range of values for the ownship heading and the range of values for the ownship velocity, we can construct linear bounds on v^x_{own} and v^y_{own}. This is done by connecting five points, a, b, c, d and e, where a and b are the points at two extreme angles and minimum velocity, c and d are the two extreme angles and max velocity, and e is the point at the intersection of the tangent lines of the maximum velocity circle at c and d. A visualization is shown in Figure 2. We use $q_\theta = 1.5$ deg (as it makes for a cleaner backreachability step), which guarantee all possible v^x_{own}, v^y_{own} values are covered.

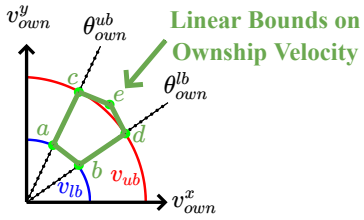


Figure 2. The ownship velocity range and heading angle range are used to create linear bounds on v^x_{own} and v^y_{own} by connecting the points a, b, c, d and e.

3.2. Backreachability from Each Partition

Once a covering of the entire set of unsafe states is performed, for each partition we compute the *exact* set of predecessor states that can lead to the states in the partition at a previous step. This process is repeated until either no predecessors exist or an initial state predecessor is found[1], as described in Definition 2.2. In the latter case, a path exists from an initial state to a partition of the unsafe states in the quantized closed-loop system. Otherwise, if no partitions contain unsafe paths, then the quantized closed-loop system is safe.

The `check_state` function in Algorithm 1 recursively computes and checks predecessors. The input is a state set $\mathcal{S}$, which is initially an 8-d partition of the unsafe states represented as an $\mathcal{AH}$-Polytope, as well as the associated value of α_{prev} and the time to loss of vertical separation, $\tau = 0$ in all unsafe states.

Algorithm 1 High-level algorithm for single partition backreachability.

Function : `check_state`, Recursively checks safety of predecessors
Input : State set: $\mathcal{S}$, Prev cmd: α_{prev}, Time to loss of vertical separation: τ
Output : Verification Result (safe or unsafe)

1 $\mathcal{P}$ = `backreach_step`($\mathcal{S}$, α_{prev}) `// state set of one-step predecessors`
2 $\tau_{\text{prev}} = \tau - \dot{\tau}$ `// $\dot{\tau}$ is fixed at either 0 or -1`
3 **for** α_{prevprev} in [COC, WL, WR, SL, SR] **do**
4 predecessor_quanta ← List()
5 all_correct ← TRUE
6 **for** q in `possible_quantized_states`($\mathcal{P}$) **do**
7 **if** `run_network`(α_{prevprev}, τ_{prev}, q) = α_{prev} **then**
8 predecessor_quanta.append(q)
9 **if** $\rho_{min}(q) > 60760$ **then**
10 **return** unsafe `// predecessor is valid initial state`
11 **else**
12 all_correct ← FALSE
13 **end**
14 **if** all_correct **then**
15 `// recursive case without splitting`
16 **if** `check_state`($\mathcal{P}$, $\alpha_{prevprev}$, τ_{prev}) = unsafe **then**
17 **return** unsafe
18 **end**
19 **else**
20 `// recursive case with splitting along quantum boundaries`
21 **for** q in predecessor_quanta **do**
22 $\mathcal{T}$ ← `quantized_to_state_set`(q)
23 $\mathcal{Q} \leftarrow \mathcal{T} \cap \mathcal{P}$
24 **if** `check_state`($\mathcal{Q}$, $\alpha_{prevprev}$, τ_{prev}) = unsafe **then**
25 **return** unsafe
26 **end**
27 **end**
28 **end**
29 **return** safe

In line 1, `backreach_step` is called, which returns the predecessor set of states as an $\mathcal{AH}$-polytope $\mathcal{P}$. This is done by taking the linear derivative matrix A_c from Equation 2 with the value of c corresponding to α_{prev}, and then computing the matrix exponential $W = e^{-A_c}$. The resulting matrix is the solution matrix for the system one second prior. A linear transformation of the $\mathcal{AH}$-polytope S is then performed by W in order to obtain $\mathcal{P}$. In line 2, the value of the time to loss of vertical separation at the previous step τ_{prev} is computed. This either always equals 0 if $\dot{\tau} = 0$

[1]Degenerate paths could theoretically exist of infinite length that never include a valid initial state, but we did not observe this occurring in practice.

for the current partition corresponding to in-plane flight, or increases by 1 at each call to `check_state` if $\dot{\tau} = -1$ for out-of-plane flight.

Next, the algorithm computes states in $\mathcal{P}$ where the command produced by the networks was α_{prev} and the time to loss of vertical separation was the value at the previous step, τ_{prev}. This requires iterating over the five possible networks that could have been used at the prior state (the loop on line 3). For each network (corresponding to α_{prevprev}), we check each quantized state in $\mathcal{P}$ (line 6).

The `possible_quantized_states` returns a list of *quantized states*, which are 5-tuples of integers, $q = (dx, dy, \theta_{own}, v_{own}, v_{int})$. The dx and dy terms correspond to the difference in positions between the intruder and ownship, divided by the position quantum q_{pos}. The θ_{own} term is the heading angle divided by q_θ, and the velocities v_{own} and v_{int} are the fixed aircraft velocities, integer divided by q_{vel}. The function computes the possible quantized states by using linear programming to find $\mathcal{P}$'s bounding box, and then looping over possible quantized states to check for feasibility when intersected with $\mathcal{AH}$-polytope $\mathcal{P}$.

Line 7 runs the neural network corresponding to α_{prevprev} on quantized state q to check if the correct command (α_{prev}) is obtained. This process requires converting from the quantized state (a 5-tuple of integers) to continuous inputs for the neural network. To do this, we use Equation 2.2, noting that the θ_{own} is quantized using q_θ, θ_{int} is always 0, and the computation of ρ and θ uses the dequantized value of dx and dy ($x_{int} - x_{own}$ is taken to be $\frac{q_{\text{pos}}}{2} + dx * q_{\text{pos}}$).

When the network output matches the required α_{prev} command, line 8 adds the quantized state to the valid list of predecessors `predecessor_quanta`. Otherwise, line 12 sets the `all_correct` flag to `false`, since some of the quantized states are not valid predecessors. Line 10 checks if the predecessor state satisfies the initial state condition, in which case an unsafe path has been found. On this line, $\rho_{\text{min}}(q)$ is the minimum aircraft separation distance in the quantized state q, which must be greater than 60760 ft in an initial state.

After classifying each quantized predecessor state, either all quantized states had the correct output or some did not. Based on this, we either recursively call `check_state` on the entire set $\mathcal{P}$ (line 16), or we split the set $\mathcal{P}$ into parts, and only recursively call `check_state` on parts that had the correct output. On line 22, `quantized_to_state_set` returns the 8-d continuous states corresponding to the quantized state q, which is then intersected with $\mathcal{P}$ before being recursively passed to `check_state`. When splitting is performed, it is possible that no states had the correct output (`predecessor_quanta` may be empty).

We next prove the described algorithm is sound with respect to the safety of the quantized closed-loop system.

Theorem 3.1 (Soundness)**.** *If* `check_state` *returns* safe *for every partition, the quantized closed-loop system is safe.*

3.3. Falsification of Original (Unquantized) System

The algorithm in the previous section can be used to efficiently find unsafe paths of the original, unquantized, closed-loop neural network control system. This is done by repeatedly calling the algorithm with smaller and smaller quantization constants q_{pos}, q_{vel} and q_θ and checking the quantized system for safety.

At each step if the safety proof fails, with small modifications to `check_state` we can get the trace corresponding to the unsafe path for each partition. In particular, rather than simply returning unsafe on line 10, we can instead return the set of unsafe initial states `quantized_to_state_set`(q) $\cap\ \mathcal{P}$. A witness point inside this set can be obtained through linear programming[2]. This witness point is then executed on the original system, without quantization, checking for safety. If the witness point is safe in the non-quantized system, the quantization constants are refined by taking turns dividing each of them in half.

Theorem 3.2 (Completeness)**.** *By following the falsification approach above and repeatedly refining q_{pos}, q_{vel} and q_θ, either we will prove the quantized system is safe or find an unsafe trace in the original, unquantized system.*

4. Evaluation

We implemented the approach and set out to prove the safety of quantized closed-loop air-to-air collision avoidance system. We ran the measurements on an Amazon Web Services (AWS) Elastic Computing Cloud (EC2) server with a `c6i.metal` instance type, which has a 3.5 GHz Intel Xeon processor with 128 virtual CPUs, and 256 GB memory. The algorithm is easily parallelized as proofs for each partition of the unsafe states can be checked independently.

4.1. Complete Proof of Safety Attempt

We first attempted a proof of safety for the entire range of unsafe states for ACAS Xu. For this, we started with large quantization values, q_{pos} = 500 ft, q_{vel} = 100 ft/sec, and q_θ = 1.5 deg. In this case, the unsafe near-mid-air collision circle of radius 500 ft can be covered with 4 partitions, the complete velocity range of the ownship $[100, 1200]$ needs 11 partitions, the velocity of the intruder $[0, 1200]$ needs 12 partitions, the heading angle of the ownship is divided

[2]For witness points, we use the Chebyshev center of the six-dimensional state polytope (removing y_{int} and v^y_{int} since they are fixed at zero), as it helps avoid numerical issues that can occur at the boundaries of the set.

into $\frac{360 \text{ deg}}{1.5 \text{ deg}}$ = 240 partitions, and there are 5 choices for the α_{prev} and two possibilities to check for $\dot{\tau}$. Multiplying these together, we get a total of 1267200 partitions of the unsafe states, each of which we pass to `check_state` (Algorithm 1).

This quickly, within a minute, finds counterexamples in the quantized system. When the witness initial states of the quantized counterexample are replayed on the original non-quantized system, according to the falsification algorithm from Section 3.3, these were also found to be unsafe! The exact runtime before an unsafe case is found depends on the order in which the partitions are searched, but we found that although changing this did affect the counterexample produced, the runtime was usually less than a minute. Two of the unsafe cases are shown in Appendix C in Figure 3 in parts (a) and (b).

In the situation shown in Figure 3(a), the intruder starts beyond the range of the network ($\rho > 60780$ ft). As soon as the intruder gets in range, a turn is commanded, but the velocity of the ownship is slow and a collision still occurs. This situation looks like it could be fixed by increasing the range of the system beyond 60780 ft—likely requiring retraining the networks—to allow a turn to be commanded earlier. Alternatively, perhaps adding a "do not turn" option as a possible output would be another way to address this scenario (clear-of-conflict could allow the ownship to maneuver as desired which may be unsafe here).

Figure 3(b) shows another unsafe case found that is particularly concerning. This is a tail-chase scenario, although the ownship is already moving away from the straight-line trajectory of the intruder. The system nonetheless commands a turn and actively maneuvers the ownship aircraft back into the path of the intruder. This situation demonstrates one of the dangers of the collision risk metric used to evaluate the effectiveness of many air-to-air collision avoidance systems, which compares the number of near mid-air collisions (NMAC) with and without the system using a large number of simulations. Although a system can be effective by this metric, in specific cases it may still create collisions that would not otherwise have occurred, as demonstrated in this scenario.

4.2. Proving Safety in More Limited Operating Conditions

As the proof of safety for the entire operating range failed, we next tried to prove safety in restricted operating conditions. Many of the unsafe situations found, including the two above, had a slow ownship velocity and a fast intruder. By making the ownship fast enough, we hypothesized collisions could be avoided.

When we restricted the range of v_{own} to be in $[1000, 1200]$ ft/sec, using q_{pos} = 250 ft, q_{vel} = 50 ft/sec, and q_θ = 1.5 deg, we were able to guarantee safety of the quantized closed-loop neural network control system. The proof requred checking 3.7 million cases and took about 32 minutes. The longest runtime for any single call to `check_state` (checking a single partition) was 63 seconds.

Reducing v_{own} further to [950, 1000] ft/sec made the quantized system unsafe. Following the falsification approach from Section 3.3, we refined the quantization parameters until we were able to find a counterexample in the original unquantized closed loop system. In this case, the ownship was moving with $v_{own} = 964.1$ ft/sec, and the time to loss of vertical separation τ was initially 75 secs (the quantized system was safe for in-plane flight, with $\dot{\tau} = 0$). This case is shown in Appendix C in Figure 3(c).

From the other side, we can alternatively attempt to prove safety under the assumption that the intruder is slow without restricting the ownship's velocity. In this case, the method also finds unsafe counterexamples in the unquantized system, such as the 159 second trace shown in Figure 3(d) with $v_{int} = 390.1$ ft/sec. In this case, the command switch from weak-left to strong-right a few seconds before the collision corresponds to the relative position angle θ wrapping from $-\pi$ to π. This discontinuity in the network input between successive steps is a strong candidate root cause of the eventual near mid-air collision.

5. Conclusion

In this work, we set out to prove the *closed-loop* safety of one of the most popular benchmarks for neural network verification methods, using a new algorithm based on state quantization and backreachability. In principle, the approach scaled sufficiently well to be able to verify the system under all valid initial states and aircraft velocities. However, the proof process instead found many unsafe scenarios where the original, unquantized system had near mid-air collisions, despite ideal assumptions on sensors and maneuvering. Compared with random simulation-based analysis, we could find counterexamples at more extreme velocities, as well as provide proofs of safety of the quantized closed-loop system in more limited scenarios.

The approach is could be attractive for certification. A system with a quantization layer behaves like a large lookup table, and the method is therefore effective on any size network with any layer type, and may even be applicable to other machine learning approaches. The trade-off of quantization is usually a small degradation in performance of the controller, with a significant benefit of reducing analysis complexity and allowing for the possibility of verification.

References

Bak, S. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pp. 19–36. Springer, 2021.

Bak, S. and Duggirala, P. S. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, 2017.

Bak, S., Tran, H.-D., and Johnson, T. T. Numerical verification of affine systems with up to a billion dimensions. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pp. 23–32, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6282-5.

Bak, S., Tran, H.-D., Hobbs, K., and Johnson, T. T. Improved geometric path enumeration for verifying relu neural networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

Bak, S., Liu, C., and Johnson, T. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

Chen, X., Ábrahám, E., and Sankaranarayanan, S. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pp. 258–263. Springer, 2013.

Clavière, A., Asselin, E., Garion, C., and Pagetti, C. Safety verification of neural network controlled systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 47–54. IEEE, 2021.

Duggirala, P. S. and Viswanathan, M. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pp. 477–494. Springer, 2016.

Dutta, S., Chen, X., and Sankaranarayanan, S. 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*, pp. 157–168, 2019.

Forets, M. and Schilling, C. Conservative time discretization: A comparative study. *arXiv preprint arXiv:2111.01454*, 2021.

Hagemann, W. Reachability analysis of hybrid systems using symbolic orthogonal projections. In *International Conference on Computer Aided Verification*, pp. 407–423. Springer, 2014.

Han, Z. and Krogh, B. H. Reachability analysis of large-scale affine systems using low-dimensional polytopes. In *International Workshop on Hybrid Systems: Computation and Control*, pp. 287–301. Springer, 2006.

Huang, C., Fan, J., Li, W., Chen, X., and Zhu, Q. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

Ivanov, R., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 169–178, 2019.

Jia, K. and Rinard, M. Verifying low-dimensional input neural networks via input quantization. In *International Static Analysis Symposium*, pp. 206–214. Springer, 2021.

Johnson, T. T., Lopez, D. M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T. J., Weimer, J., and Lee, I. Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. *EPiC Series in Computing*, 80:90–119, 2021.

Julian, K. D. and Kochenderfer, M. J. Guaranteeing safety for neural network-based aircraft collision avoidance systems. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pp. 1–10. IEEE, 2019.

Julian, K. D., Lopez, J., Brush, J. S., Owen, M. P., and Kochenderfer, M. J. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–10. IEEE, 2016.

Julian, K. D., Kochenderfer, M. J., and Owen, M. P. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, 2019.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Kochenderfer, M. J. and Chryssanthacopoulos, J. Robust airborne collision avoidance through dynamic programming. *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371*, 130, 2011.

Kochenderfer, M. J., Edwards, M. W., Espindle, L. P., Kuchar, J. K., and Griffith, J. D. Airspace encounter

models for estimating collision risk. *Journal of Guidance, Control, and Dynamics*, 33(2):487–499, 2010.

Liu, C., Arnon, T., Lazarus, C., Barrett, C., and Kochenderfer, M. J. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.

Lopez, D. M., Johnson, T. T., Tran, H.-D., Bak, S., Chen, X., and Hobbs, K. Verification of neural network compression of acas xu lookup tables with star set reachability. In *AIAA Scitech 2021 Forum*. AIAA, January 2021.

Marston, M. and Baca, G. ACAS-Xu initial self-separation flight tests. http://hdl.handle.net/2060/20150008347, 2015.

Olson, W. A. Airborne collision avoidance system x. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 2015.

Sadraddini, S. and Tedrake, R. Linear encodings for polytope containment problems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4367–4372. IEEE, 2019.

Scott, J. K., Raimondo, D. M., Marseglia, G. R., and Braatz, R. D. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126–136, 2016.

Tran, H.-D., Cai, F., Diego, M. L., Musau, P., Johnson, T. T., and Koutsoukos, X. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s): 1–22, 2019a.

Tran, H.-D., Lopez, D. M., Musau, P., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*, pp. 670–686. Springer, 2019b.

Tran, H.-D., Xiang, W., and Johnson, T. T. Verification approaches for learning-enabled autonomous cyber-physical systems. *IEEE Design & Test*, 2020a.

Tran, H.-D., Yang, X., Manzanas, D., Musau, P., Nguyen, L., Xiang, W., Bak, S., and Johnson, T. T. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020b.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium*, pp. 1599–1614, 2018.

Zombori, D., Bánhelyi, B., Csendes, T., Megyeri, I., and Jelasity, M. Fooling a complete neural network verifier. In *International Conference on Learning Representations*, 2020.

A. $\mathcal{AH}$-Polytope

Definition A.1 ($\mathcal{AH}$-Polytope)**.** An $\mathcal{AH}$-Polytope is a tuple $\Theta = \langle V, c, C, d\rangle$ that represents a set of states as follows:

$$[\![\Theta]\!] = \{x \in \mathbb{R}^n \mid \exists \alpha \in \mathbb{R}^m,\ x = V\alpha + c \wedge C\alpha \leq d\}.$$

Proposition A.2 (Affine Mapping)**.** *An affine mapping of an $\mathcal{AH}$-Polytope $\Theta = \langle V, c, C, d\rangle$ with a mapping matrix W and an offset vector b is a new $\mathcal{AH}$-Polytope $\Theta' = \langle V', c', C', d'\rangle$ in which $V' = WV,\ c' = Wc + b,\ C' = C,\ d' = d$.*

Proposition A.3 (Linear Transformation)**.** *A linear transformation of an $\mathcal{AH}$-Polytope with a matrix W is an affine mapping using mapping matrix W and an offset vector of $b = 0$.*

Proposition A.4 (Intersection)**.** *The intersection of $\Theta = \langle V, c, C, d\rangle$ and a half-space $\mathcal{H} = \{x \mid Gx \leq g\}$ is a new $\mathcal{AH}$-Polytope $\Theta' = \langle V', c', C', d'\rangle$ with $c' = c,\ V' = V,\ C' = [C; GV],\ d' = [d; g - Gc]$.*

Proposition A.5 (Linear Optimization)**.** *Linear optimization in given a direction $w \in \mathbb{R}^n$ over a star set $\Theta = \langle V, c, C, d\rangle$ can be solved with linear programming as follows:* $\min(w^T x),\ s.t.\ x \in \Theta = w^T c + \min(w^T V\alpha),\ s.t.\ C\alpha \leq d$.

B. Proofs

B.1. Proof for Theorem 3.1

Proof. We proceed by contraction. Assume the quantized closed-loop system is unsafe and so these exists a finite path from an initial state to an unsafe state, $s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_{n-1}} s_n$. Since the unsafe state partitioning covers the full set of unsafe states, the unsafe state s_n is in some partition. We can follow the progress of $s_n \in \mathcal{S}$, through `check_state` at each recursive call.

At each call, $s_i \in \mathcal{S}$ has a predecessor $s_{i-1} \in \mathcal{P}$ that gets to s_i using command α_{i-1}. In the call to `check_state`, α_{prev} will be α_{i-1}. The value of τ_{prev} is incremented at each call on line 2 and so always correctly corresponds to s_{i-1}. Since $s_{i-1} \in \mathcal{P}$, s_{i-1} will also be in one of the quantized states q_{i-1} checked on line 6. The existence of the counterexample path segment $\xrightarrow{\alpha_{i-2}} s_{i-1} \xrightarrow{\alpha_{i-1}} s_i$ means that the condition on line 7 will be true when $\alpha_{\text{prevprev}} = \alpha_{i-2}$, and so q_{i-1} will be added to `predecessor_quanta`. Since s_{i-1} is both in $\mathcal{P}$ and in the state set corresponding to a quantized state in `predecessor_quanta`, it will be used in a recursive call to `check_state`. This argument can be repeated for all states in the unsafe path back to the initial state s_1, which would have been returned as unsafe on line 10 rather than used in a recursive call. This contradicts the assumption that `check_state` returned safe for every partition. □ □

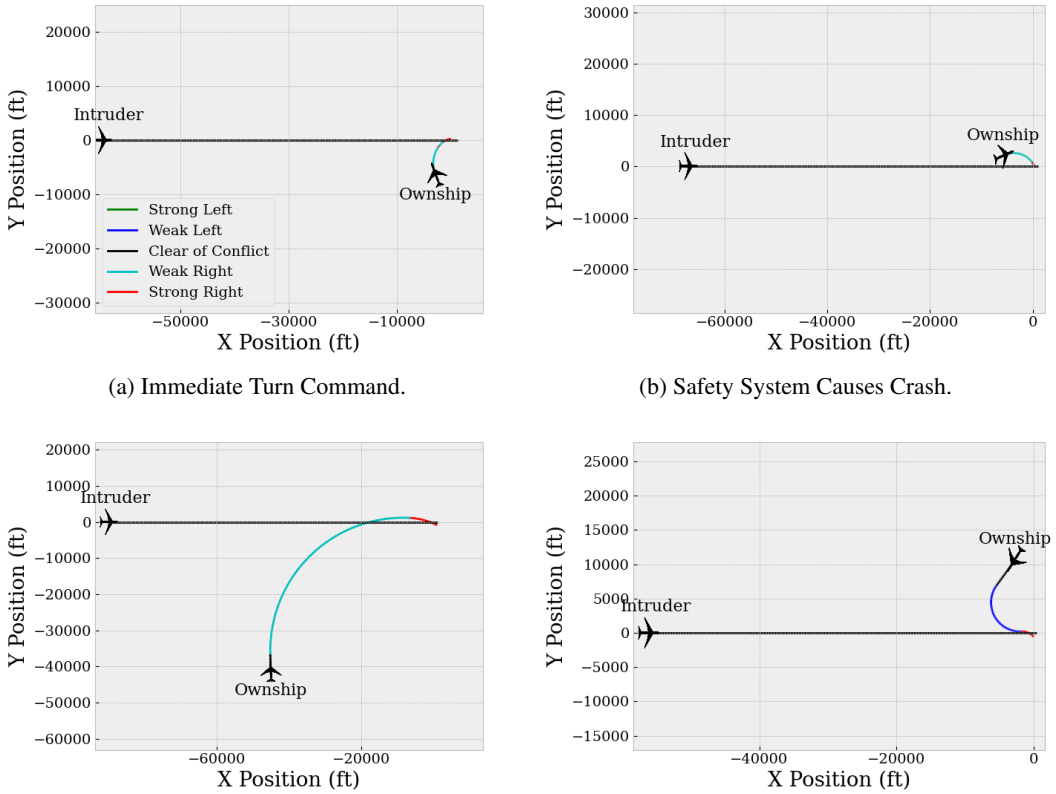
B.2. Proof for Theorem 3.2

Proof. First, consider the case that the system is *robustly unsafe*, which we define as there existing a ball $\mathcal{B}_{\text{init}}$ of initial states of radius $\delta > 0$ that all follow the same command sequence $\alpha_1, \alpha_2, \ldots, \alpha_n$ and end in the unsafe set. Since all the initial states follow the same command sequence, the linear transformations corresponding to the commands $\alpha_1, \alpha_2, \ldots, \alpha_n$, which we call $A_{c_1}, A_{c_2}, \ldots, A_{c_n}$ can be multiplied together into a single matrix that transforms initial states to unsafe states, $A_C = A_{c_n} \ldots A_{c_2} A_{c_1}$. The matrix A_C is invertible since all the transformations corresponding to each command $A_{c_1}, A_{c_2}, \ldots, A_{c_n}$ are invertible. The matrix A_C being invertible means that since the volume of the ball in the initial states $\mathcal{B}_{\text{init}}$ is nonzero, the corresponding set of states in the unsafe set is an ellipsoid with nonzero volume, which we call $\mathcal{E}_{\text{unsafe}}$. Through refinement of the quantization parameters q_{pos}, q_{vel} and q_θ, eventually a partition will be entirely contained in $\mathcal{E}_{\text{unsafe}}$. When this happens, every witness point of the quantized counterexample from that partition will be in $\mathcal{B}_{\text{init}}$, and so will be an initial state of an unsafe oath of the original, unquantized system.

Perhaps less practically, even if the original system is not robustly unsafe, the process still will theoretically terminate when finite-precision numbers are used in the non-quantized system, such as with air-to-air collision avoidance neural networks that use 32-bit floats. As the quantization values are halved, the difference between the unsafe state in the quantized and nonquantized system is also reduced, until it reaches numeric precision. □ □

C. Unsafe Encounter Images

Figure 3 shows images of unsafe encounters found using the described method.



(a) Immediate Turn Command.

(b) Safety System Causes Crash.

(c) Fast Ownship and $\tau > 0$.

(d) Slow Intruder.

Figure 3. Unsafe counterexamples found in the original non-quantized NNCS.

D. Comparison with Other Approaches

As far as we are aware, the proposed method is the first to provide safety guarantees while varying all of the operating conditions of the neural network compression of the collision avoidance system.

One related technique, based on computing discrete abstractions and forward reachability was able to provide safety guarantees for the similar Horizontal CAS (Julian & Kochenderfer, 2019). This system is simpler to analyze: the inputs were modified to take in Cartesian state variables, the operating range was smaller ($\rho < 50000$), there were fewer neural networks in the system, each of which had half as many neurons per layer, and critically, fixed velocities of $v_{own} = 200$ and $v_{int} = 185$ were considered, rather than using velocity ranges. Despite these simplifications, analysis took 227 CPU hours, mostly on the neural network analysis step to analyze 74 million partitions. For a comparison, we analyzed the larger neural networks in this work with the proposed state quantization and backreachability method, using the same fixed v_{own} and v_{int} values. Using a quantized system with $q_{pos} = 250$ ft and q_θ = 1.5 deg, the method proved safety of all 38400 partitions of the unsafe states in 60.6 seconds. Also note that while the Horizontal CAS discrete abstraction approach can sometimes prove safety, it would be poor at generating counterexamples, as abstract reachability overapproximates the true reachable set; abstract counterexamples do not correspond to real counterexamples. In contrast, the backwards reachability performed in Algorithm 1 is exact with respect to the quantized system, and the gap between the quantized and original system can be reduced by refining the quantization parameters, making it highly effective for counterexample generation.

We also compared our method with simulation-based analysis, which cannot provide guarantees about system safety but should be able to find unsafe counterexamples if enough simulations are attempted, as the system was shown to be unsafe. In earlier work (Julian et al., 2019), 1.5 million encounters were simulated for the original neural network compression to evaluate the risk of collisions, sampling from probability distributions of actual maneuvers and taking into account sensor noise. We evaluated the same number of simulations without sensor noise and sampling over the entire set of operating conditions, in order to match the assumptions used in the safety proof. We generated uniform random initial states by

considering an initial $\rho \in [60760, 63160]$ and θ, ψ, v_{own} and v_{int} in their entire operating range. When considering $\dot{\tau} = -1$, we assigned the initial value of τ between 25 and 160 seconds, as the unsafe case in Figure 3(d) was a 159 second trace. We repeated the process of running 1.5 million simulations one hundred times each for both $\dot{\tau} = -1$ and $\dot{\tau} = 0$, in order to account for statistical noise.

In the $\dot{\tau} = 0$ case, each batch of 1.5 million simulations found on average 17.07 unsafe paths. The unsafe cases were dominated by situations where the intruder velocity was low and the ownship velocity was high. The mean value of v_{int} was 997.8, with a standard deviation of 147.5. The lowest values of v_{int} over the unsafe cases in all 150 million simulations was 927.6, whereas Figure 3(d) showed a case with $v_{int} = 390.1$ found with our approach. The mean value of v_{own} in the unsafe cases was 133.4 with a standard deviation of 43.0. The greatest value of v_{own} over all the unsafe cases found with 150 million simulations was 452.3, whereas our approach found an in-plane case with $v_{own} = 881.6$.

The performance of simulation analysis for the out-of-plane case is even worse, as the initial state must also correctly choose the value of the time to loss of vertical separation τ in order to find a collision. Each batch of 1.5 million simulations with $\dot{\tau} = -1$ had on average 0.07 unsafe simulations. The maximum ownship velocity v_{own} in the unsafe cases had a mean of 175.4 with a standard deviation of 77.9. The greatest value of v_{own} over the unsafe cases found in all 150 million simulations was 343.0, whereas our approach found a case with $v_{own} = 964.1$, as shown before in Figure 3(c).

Overall, while simulation analysis may find some unsafe cases, it would be difficult to find the extreme velocity cases discovered with the proposed approach. Further, simulation analysis is incomplete and cannot prove safety for the system under subsets of operating conditions as was done in Section 4.2.

E. Related Work

Simulation-based Safety Analysis. The air-to-air collision avoidance system was originally evaluated using 1.5 million simulations (Kochenderfer et al., 2010) based on Bayesian statistical encounter models. This uses relaxed assumptions compared with our work, such as allowing for changes in acceleration. The output of such analysis is not a yes/no assessment of safety, as the system can clearly be unsafe if the intruder is faster than the ownship and maneuvers adversarially, but rather a risk score assessment of the change in safety compared to without using the system. Via simulation, given a bounded uncertainty in sensing and control, the probability of near-mid-air-collision was about 10^{-4} (Julian et al., 2019). Although simulations show that the system may be unsafe, we do not know if the collision occurs due to the uncertainty or the system itself. In this work, we could show that the system itself was unsafe, even if we have perfect sensing and control.

Verification of NNCS. The Verisig approach (Ivanov et al., 2019) verifies a NNCS by transforming a network with a sigmoidal neural network controller to an equivalent hybrid system that can be analyzed with Flow* (Chen et al., 2013), a well-known tool for verifying nonlinear hybrid systems. Another method (Huang et al., 2019; Dutta et al., 2019) combines polynomial approximation of the neural network controller with the plant's physical dynamics to construct a tight overapproximation of the system's reachable set. The star set approach (Tran et al., 2019a) shows that the exact reachable set of an NNCS with a linear plant model and a ReLU neural network controller can be computed, although this is expensive when initial states are large. These methods build upon open-loop neural network verification algorithms (Liu et al., 2019; Tran et al., 2020a), which can be difficult to scale to large complex networks (Bak et al., 2021) and can sometimes lose soundness due to floating-point numeric issues (Zombori et al., 2020). The proposed quantization analysis only needs to execute neural networks, and so does not suffer from these problems.

Verification of the Closed-loop Air-to-Air Collision Avoidance System. Existing works have verified NNCS with a single neural network controller on a small set of initial states (Johnson et al., 2021). The closed-loop system involves switching between multiple neural networks and has a large set of initial states, creating a unique challenge for verification. The simplified Horizontal CAS system was analyzed using fast symbolic interval analysis for neural network controllers (Wang et al., 2018) to construct a discrete abstraction (Julian & Kochenderfer, 2019). This method can consider sensor uncertainty, inexact turn commands, and pilot delay, although simplified assumptions are made, as discussed in Section D. Recently, the same system as this work has been verified with extensions of the symbolic interval method (Clavière et al., 2021) and with star-based reachability (Lopez et al., 2021) in nnv (Tran et al., 2020b) and nnenum (Bak, 2021). These approaches use forward reachability analysis and provide sound but not complete verification results. However, verification has only been demonstrated for specific scenarios with small sets of initial states, not the full operating conditions considered here.