Towards Verification of Neural Networks for Small Unmanned Aircraft Collision Avoidance

Ahmed Irfan
Stanford University
Stanford, CA, USA
irfan@cs.stanford.edu

Kyle D. Julian
Stanford University
Stanford, CA, USA
kjulian3@stanford.edu

Haoze Wu
Stanford University
Stanford, CA, USA
haozewu@stanford.edu

Clark Barrett
Stanford University
Stanford, CA, USA
barrett@cs.stanford.edu

Mykel J. Kochenderfer
Stanford University
Stanford, CA, USA
mykel@stanford.edu

Baoluo Meng
GE Global Research
Niskayuna, NY, USA
baoluo.meng@ge.com

James Lopez
GE Global Research
Niskayuna, NY, USA
lopezj@ge.com

***Abstract*—The ACAS X family of aircraft collision avoidance systems uses large numeric lookup tables to make decisions. Recent work used a deep neural network to approximate and compress a collision avoidance table, and simulations showed that the neural network performance was comparable to the original table. Consequently, neural network representations are being explored for use on small aircraft with limited storage capacity. However, the black-box nature of deep neural networks raises safety concerns because simulation results are not exhaustive. This work takes steps towards addressing these concerns by applying formal methods to analyze the behavior of collision avoidance neural networks both in isolation and in a closed-loop system. We evaluate our approach on a specific set of collision avoidance networks and show that even though the networks are not always locally robust, their closed-loop behavior ensures that they will not reach an unsafe (collision) state.**

***Index Terms*—Airborne Collision Avoidance, Deep Neural Network, Formal Methods, Local Robustness, Reachability.**

I. Introduction

The Federal Aviation Administration (FAA) is leading the development of a family of collision avoidance systems for unmanned aircraft, including ACAS Xu for large unmanned aircraft and the derived variant for sUAS called ACAS sXu [1]. ACAS sXu, like other ACAS systems, uses numeric lookup tables optimized offline for decision making. However, due to the large size of the table and the limited memory availability on sUAS, a compressed representation of the table is desired. Recent work [2] showed that a deep neural network (DNN) approximation of the table can reduce the needed memory by a factor of 1000. The work also demonstrated through simulation that the DNN representation does not increase computation time or diminish operational performance. The tabular version of ACAS sXu has been integrated and flight tested [3], and there are plans to do the same with the DNN version.

DNNs are known for their opaqueness and susceptibility to adversarial attacks [4], [5]. The black-box nature of DNNs calls into question their trustworthiness and hinders their application to safety-critical domains. In this work, we take steps towards addressing safety concerns by applying formal methods.

We train DNNs from a lookup table of an early prototype of ACAS sXu (the official table may be obtained from the ASTM or RTCA standards organizations). We explore the verification of DNNs in a closed-loop system that captures the dynamics of an aircraft and uses a DNN controller for advisories. We also explore verifying the neural networks in isolation, by analyzing the local robustness of the DNN with respect to the original table. Local robustness requires that there be no perturbation within a small range in the network's inputs that could cause a significant difference in the network's output. To do the analysis, we first cluster the input regions with the same output label into hypercubes and create verification queries for each cluster. Then, we check the queries using a neural network verification tool called Marabou [6].

We also analyze the DNNs within a closed-loop system to evaluate whether the network can reach an unsafe (collision) state. To analyze the closed-loop system, we adapt a reachability method [7] that was originally proposed for analyzing a vertical collision avoidance system. In that work, the neural network outputs were bounded using the neural network verification tools Reluplex [8] and Reluval [9]. The reachability method determines all possible ways aircraft encounters can resolve under certain assumptions about the aircraft dynamics. We apply the method for the horizontal component of the ACAS sXu system. The reachability analysis shows that even though the neural networks are not locally robust everywhere, they are still safe in the sense that an aircraft following advisories from the network will avoid collision.

The paper is organized as follows: We first describe the table of a collision avoidance prototype and its neural network representation in Sec. II and Sec. III, respectively. We then discuss our approach to local robustness verification and present our experimental results in Sec. IV. In Sec. V, we present the dynamics of the closed-loop system; we overview the reachability method that we use for the verification of the closed-loop system and present the experimental results on applying the reachability method on the system. We conclude our work and discuss future directions in Sec. VI.

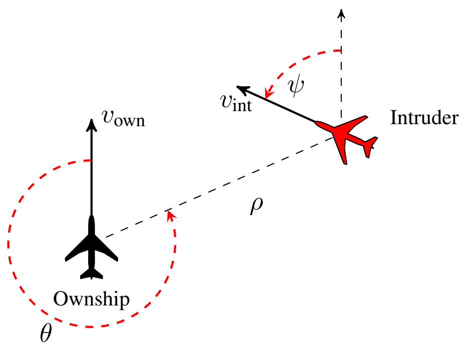


Fig. 1. Aircraft encounter geometry for ACAS sXu.

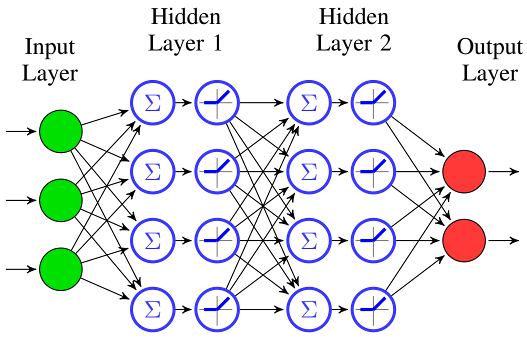


Fig. 2. An example of a deep neural network.

II. ACAS sXu Prototype

ACAS sXu [1] is a variant of the ACAS X aircraft collision avoidance family designed to help small unmanned aircraft avoid near midair collisions (NMACs) with other manned aircraft or large UAS. Like other ACAS X systems, ACAS sXu contains logic to compute the encounter state from sensor information and to issue maneuver advisories when needed to avoid an NMAC. Advisories are decided using tables of advisory scores for a discrete set of encounter states that covers the full range of possible encounter states. These tables are optimized offline by modeling the problem as a Markov decision process and solving it with dynamic programming [10]. ACAS sXu uses the encounter state computed from sensor information to compute the best scoring advisories using nearest-neighbor interpolation. ACAS sXu contains separate tables for vertical and horizontal maneuvering, along with logic for deciding which type of maneuver to issue, but this work will focus solely on the horizontal table.

The horizontal logic table provides the scores of discrete state-action pairs, where a state is a seven-dimensional vector, and an action is one of the horizontal turning advisories. The seven state variables describe the encounter with an intruder aircraft (Table I). The first five variables define the horizontal geometry and are shown in Fig. 1. The τ variable is a countdown to the time when vertical separation will be lost, so the aircraft must be separated horizontally to avoid an NMAC, which is defined as separation less than 500 ft horizontally when vertical separation is lost. Additionally, including the previous advisory, s_{adv}, allows the system to alert consistently.

TABLE I
ACAS sXu state variables

Variable	Description	Values	Num
ρ (ft)	Range to intruder	$[499, 36656]$	20
θ (rad)	Bearing angle to intruder	$[-\pi, \pi]$	41
ψ (rad)	Relative heading angle of int.	$[-\pi, \pi]$	41
v_{own} (ft/s)	Ownship speed	$[100, 472]$	6
v_{int} (ft/s)	Intruder speed	$[0, 1200]$	12
τ (s)	Time to loss of vert. separation	$[0, 101]$	10
s_{adv}	Previous advisory	COC, WL, WR SL, SR	5

Each state variable has a range of values that are discretized. The discretization scheme in Table 1 results in 121 million possible states.

The action space consists of five turning advisories of different strengths and directions. The advisories are Clear of Conflict (COC), Weak Left (WL), Weak Right (WR), Strong Left (SL), and Strong Right (SR). Weak advisories represent $1.5\,°/\text{s}$ turns, and strong advisories represent $3.0\,°/\text{s}$. COC allows the ownship to maneuver freely. In total, the table stores values for every state-action pair, resulting in a table occupying approximately 2 GB with 32-bit floating point precision.

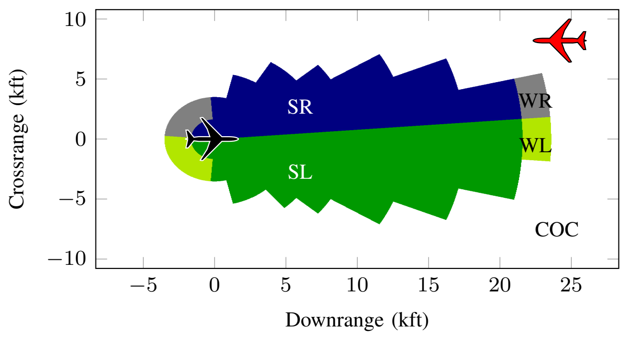
III. Neural Network Approximation

The size of the horizontal logic table can be too large for use in certified or small avionics systems with limited memory. A neural network can be trained to approximate the logic table and significantly reduce required storage space [2]. Previous work showed that a neural network representation maintains accuracy while significantly reducing representation size and outperforming other compression methods such as decision trees or symmetry analysis [2], [11].

A feed-forward *deep neural network* (DNN) consists of a sequence of layers, including an input layer, an output layer, and one or more hidden layers in between. A DNN example is shown in Fig. 2. Each non-input layer contains *neurons*, whose values are computed as a weighted sum of the outputs of the preceding layer plus a bias term. For neurons in the hidden layers, the weighted sum and bias values are passed through a non-linear function, known as an *activation function*. A common activation function is the Rectified Linear Unit (ReLU), defined as $ReLU(x) = \max(0, x)$ (see [12]–[14]). In this work, we use ReLU-based DNNs.

TABLE II
Input variables of ACAS sXu prototype networks

Variable	Description	Range of values
x (ft)	Relative downrange position of intruder	$[-36656, 36656]$
y (ft)	Relative crossrange position of intruder	$[-36656, 36656]$
ψ (rad)	Relative heading angle of intruder	$[-\pi, \pi]$
v_{own} (ft/s)	Ownship speed	$[100, 472]$
v_{int} (ft/s)	Intruder speed	$[0, 1200]$



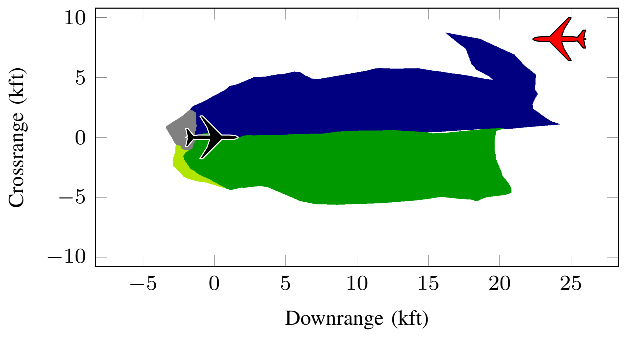


Fig. 3. Policy of table (top) and DNN (bottom) representations for a head-on encounter.

We trained 50 different neural networks, one for each combination of s_{adv} and $\tau \in \{0, 1, 5, 10, 20, 40, 60, 80, 100, 101\}$. A set of smaller networks was trained rather than a single large network to reduce runtime to evaluate the networks [2]. For each network, we used the following DNN architecture: 5 inputs, 5 outputs, and 5 hidden layers each containing 30 neurons. The inputs to each network are the five remaining state variables, but the polar variables ρ and θ were converted to Cartesian coordinates x and y via $x = \rho \cos \theta$ and $y = \rho \sin \theta$. Using Cartesian coordinates is important when analyzing the closed-loop system, which is discussed in Section V-B. The input variables of the 50 neural networks and their expected range of values is shown in Table II. The outputs of the networks are estimates of the five advisory scores. Because the scores are costs, the minimum scoring advisory is the best advisory.

Each network was trained for 200 epochs with a batch size of 512 and the Adam gradient descent method [15]. The loss function was an asymmetric mean squared error designed to train a network that accurately predicts both the scores of the advisories and the policy [2]. The trained neural networks predict the advisory scores with a root mean squared error of 3.69 (the scores have a range of 424), and the policy is correct in 94.4% of the states. In total, the 50 network representation requires 792 kB of memory using 32-bit floating point precision, which is a $2600\times$ reduction in representation size.

Fig. 3 shows a slice of the policy produced by the original table and neural network representations, where the ownship is at the origin, and each pixel is colored according to the advisory if the intruder was located at that point and heading in the direction shown by the aircraft in the upper right corner. The policy uses nearest neighbor interpolation, but the neural network is a continuous function. Visually, the neural network preserves many features of the policy, but there are some differences. These differences will be explored using neural network verification tools in the remainder of this paper.

IV. Local Robustness

When a DNN is employed in a safety-critical context, it is essential to have confidence that it will not behave in a way that will make the system unsafe. We can test the DNN on many different input points and check whether its output is safe within the context of the system. While this does increase confidence in the system, it is not sufficient to prove safety. To further increase the confidence in the system, we can show that the DNNs are not only safe on a finite set of test points, but also safe in some neighborhood of those test points. This property is known as *adversarial robustness* and can be checked with a DNN verification tool. Local adversarial robustness expresses the requirement that the network behave smoothly, i.e. that small input perturbations should not cause major spikes in the output. Because DNNs are trained over a finite set of inputs and outputs, the robustness check captures our desire to ensure that the network behaves well on inputs absent from the training and test sets. A definition [8], [16], [17] of local adversarial robustness is as follows:

Definition 1: A DNN N is δ-locally robust at input point x, iff $\forall x', \|x - x'\| \leq \delta \Rightarrow \text{label}(N, x) = \text{label}(N, x')$.

Intuitively, Definition 1 states that for input x' that is very close to x, the network assigns to x' the same label that it assigns to x; local thus refers to a local neighborhood around the point x. Larger values of δ imply larger neighborhoods, and hence better robustness. In the context of image recognition, for example, δ-local-robustness can capture the fact that slight perturbations of the input image should not result in a change of label.

By using a sufficiently fine grid of test points in the input domain and showing local robustness in a region at least as large as the distance between test points, we could increase our confidence that the network is likely to behave as intended. A natural representative set of test points is the training set because these points are considered ground truth and cover the input region where the network is expected to perform well. However, there are two difficulties with this approach:

1) *Computational cost*: Since the training data for each network contains more than 810,000 points, it could be computationally expensive to check all points for all networks individually;
2) *Decision boundaries*: For points near the decision boundary (i.e., the hypersurface that partitions the input space into sets corresponding to different output labels), we should not expect the local adversarial robustness to hold. Therefore, the parts of the input regions that are near decision boundaries should be handled differently.

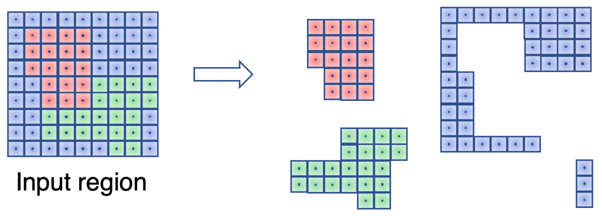


Fig. 4. Step 1 of the hypercube approach.

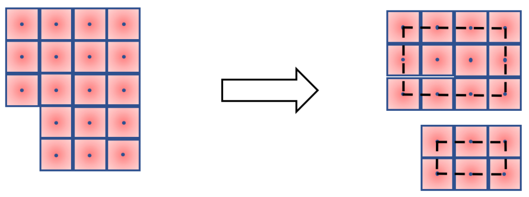


Fig. 5. Step 2 of the hypercube approach.

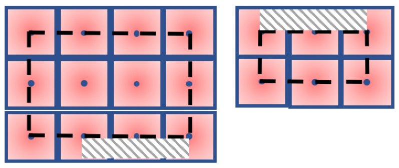


Fig. 6. Input regions excluded by the hypercube analysis.

We propose a technique that allows us to check multiple points simultaneously. The training data R (a set of points) is partitioned into disjoint subsets such that each subset S_i contains geometrically contiguous points with the same output label. Instead of individually checking robustness around each point in R, our idea is to check points simultaneously in S_i. More precisely, for each S_i, we check robustness in a region H containing only points in S_i.

Since DNN verification tools typically require that local adversarial robustness queries are given over a convex affine input region, it is important that H be symbolically represented as a convex affine set. In this work, we fix the shape of H to be a hypercube defined by lower- and upper-bounds of each input variable.

Definition 2: A DNN N is locally robust in hypercube H, iff $\forall x \in H \Rightarrow \text{label}(N, x) = \text{label}(H)$.
Here, we abuse notation and denote $\text{label}(H)$ as the label of the test points in H. Note that if adversarial examples are found in H, then by Definition 2, N is not locally robust in H. However, this result can lead to a lower estimate of the network's overall local robustness. To improve this situation, we identify subsets of H where the neural network *is* adversarially robust, allowing us to better estimate the accumulated local robustness.

To address the second challenge, we exclude the decision-boundary regions from each hypercube symbolic representation as discussed in the rest of this section.

A. A Hypercube Approach

We describe our approach, which is broken down into three steps, to establish the adversarial robustness of the trained networks.

Step 1: Decompose the training points into clusters of adjacent points with the same output label (Figure 4 is an illustration of this step in 2 dimensions);

Step 2: Decompose the points in the same cluster into sets of points such that they can be symbolically represented by a hypercube, as illustrated by Figure 5;

Step 3: Compute the volume of adversarially robust regions in each hypercube.

In Step 2, we use a greedy algorithm described in Algorithm 1 to decompose a cluster into sets of points. Given a cluster of points C, we iterate through each point p in C and check to see if it can be included in some existing set. The function $\text{extend}(hc, p)$ returns the set of training points in the minimal hypercube that contains $hc \cup \{p\}$. A point can be included in a set hc, if $\text{extend}(hc, p)$ does not include points outside C. If no existing set can be extended, then we augment hc with a new set $\{p\}$.

We symbolically represent each set of points with a minimal hypercube containing that set (regions bounded by the dotted lines in Fig. 5). This way of generating hypercubes effectively leaves out decision boundaries. However, this also leaves out some regions that are not expected to be decision boundaries (e.g., the shaded areas in Fig. 6). We can do a second analysis pass to identify those left-out regions and add them to appropriate hypercubes. However, in this work, we do not perform this second pass and leave it as future work. Using the hypercube clusters, we can check the adversarial robustness of a network in each hypercube instead of checking single points individually. This reduces the number of solver calls.

Note that in Step 3, we do not just check whether a hypercube H contains adversarial examples, because adversarial examples may only exist in small subsets of H, and concluding that H as a whole is not adversarially robust in such cases would be misleading. Therefore, in such cases, we attempt to identify hypercubes contained in H that are adversarially robust. We compute the ratio of their volume to the volume of H, a metric that we call *robust volume ratio*. If no adversarial examples exist in H, the robust volume ratio of H is 100%. With this metric, we obtain a better estimate of how robust

Algorithm 1 Decomposing a cluster of points

```
Input: A cluster of points $C$
Output: Sets of points $hcs$
$hc \leftarrow \emptyset$
while $C.\text{notEmpty}()$
  $p \leftarrow C.\text{pop}()$
  for $hc \in hcs$
    $hc' \leftarrow \text{extend}(hc, p)$
    if $hc' \subseteq C$
      $hc, C \leftarrow hc', (C \setminus hc')$
      break
  $hcs \leftarrow hcs \cup \{\{p\}\}$
return $hcs$
```

the network is in H.

B. Computing Robust Volume Ratio

Algorithm 2 shows how we compute the robust volume ratio of the hypercube H. We now describe the algorithm in detail. The set of hypercubes whose adversarial robustness to be checked is represented by Q, which is initialized with $\{H\}$. The set A contains hypercubes that are not adversarially robust and is empty in the beginning. Then, while Q is not empty, we pop a hypercube h from Q and check its adversarial robustness (this is done using the **checkRobustness** method). If h is robust, we continue to check other hypercubes in Q. Otherwise, we partition h into K disjoint hypercubes and add them to Q. Note that if the volume of h is below a certain predefined threshold — captured by the **volumeThresholdReached** method — then we add it to the set of unrobust hypercubes A. The algorithm terminates when Q becomes empty. We could then compute the total volume v_A of A, and subtract it from the volume of v_H to obtain the volume of robust regions in v_H.

In Algorithm 3, we present the **checkRobustness** method, which returns true if the given network N is robust in the hypercube H and returns false otherwise. The main part of the algorithm checks whether it is possible to have a mislabeled output. By a mislabeled output, we mean that an output variable, corresponding to a label other than the expected label l, is given a lower score than the score corresponding to l. This check is done using the **checkSat** method, which takes as input the network N and a query ϕ. The **checkSat** method invokes a DNN verification tool and returns SAT if it finds an assignment to the inputs of N that makes ϕ true; otherwise, it returns UNSAT. A SAT result means that there exists a counterexample, so in that case **checkRobustness** returns false. If for all the labels other than l, the result of **checkSat** is UNSAT, then **checkRobustness** returns true.

C. Cartesian Approximations of Polar Coordinate Rectangles

The first two inputs to the neural networks represent the relative position of the intruder with respect to the ownship and are given in rectangular coordinates. However, the hypercubes are created using the original lookup table with polar coordinates. We approximate the area defined by the first two inputs (in polar coordinates) using linear constraints in rectangular coordinates.

A rectangle in polar coordinates is defined as: i) $\rho_{\min} \le \rho \le \rho_{\max}$, ii) $\theta_{\min} \le \theta \le \theta_{\max}$. This is the region defined by the black lines in Figure 7. We can approximate this region using four linear bounds in Cartesian (rectangular) coordinates (x, y), where $x = \rho \cos\theta$ and $y = \rho \sin\theta$. We state without proof the symbolic representation of the linear approximation in rectangular coordinates (the region defined by the dashed lines in Figure 7).

The upper and lower bounds on θ are converted to linear inequalities in Cartesian coordinates using $\frac{y}{x} \ge \tan(\theta_{\min})$ and $\frac{y}{x} \le \tan(\theta_{\max})$, which is an exact representation of the θ bounds. Representing bounds on ρ requires non-linear bounds in Cartesian coordinates, so linear approximations are used, as depicted in Fig. 7. An inner approximation is used so that any counterexample found within the linear region will be a valid counterexample. The upper bound on ρ is approximated by connecting the endpoints of the bounding arc, and the lower bound on ρ is approximated using a tangent line at the midpoint of the arc. Defining $\bar{\theta} = \frac{1}{2}(\theta_{\min} + \theta_{\max})$, the equations for these boundary lines are

$$y = -\frac{1}{\tan\bar{\theta}}(x - \rho_{\max}\cos(\theta_{\min})) + \rho_{\max}\sin(\theta_{\min}) \quad (1)$$

$$y = -\frac{1}{\tan\bar{\theta}}(x - \rho_{\min}\cos(\bar{\theta})) + \rho_{\min}\sin(\bar{\theta}), \quad (2)$$

and the direction of the bounds can be determined by using the test point $(0, 0)$. For Eq. 1, the linear approximation of the $\rho_{\max}$ bound, $(0, 0)$ should always be a point that satisfies the constraint. For Eq. 2, the linear approximation of the $\rho_{\min}$ bound, $(0, 0)$ is a point that should not satisfy the constraint.

D. Experiments

We performed the analysis on the ACAS sXu prototype networks to establish their adversarial robustness. We excluded from our analysis the networks which correspond to $\tau = 101$ (5 out of 50 networks), because their training data gives a uniform advisory of COC and those networks can be replaced with a constant. As a proof of concept, for the remaining 45 networks, we randomly sampled about 1% of the hypercubes and computed their robust volume ratio. We ran our main experiment on a cluster equipped with Intel Xeon E5-2620 v4 CPUs running Ubuntu 16.04.

Algorithm 2 Compute Robust Volume Ratio

```
Input: A neural network $N$, a hypercube $H$
Output: the robust volume ratio of $H$, $r$
$Q \leftarrow \{H\}$
$A \leftarrow \emptyset$
while $Q.\mathrm{notEmpty}()$
    $h \leftarrow Q.\mathrm{dequeue}()$
    $isRobust \leftarrow \mathbf{checkRobustness}(N, h)$
    if $\neg isRobust$
        if $\mathbf{volumeThresholdReached}(h)$
            $A.\mathrm{enqueue}(h)$
        else
            for $h' \in \mathrm{partition}(h, K)$
                $Q.\mathrm{enqueue}(h')$
$v_H = \mathbf{volume}(H)$
$v_A = \sum_{h \in A} \mathbf{volume}(h)$
return $(v_H - v_A)/v_H$
```

Algorithm 3 checkRobustness

```
Input: A neural network $N$, a hypercube $H$
Output: true if no adversarial examples were found in $H$, false otherwise.
$l \leftarrow \mathrm{label}(H)$
for $l' \in AllLabels$
    if $l' \neq l$ & $\mathbf{checkSat}(N, H \wedge (y_{l'} \le y_l)) = \mathtt{SAT}$
        return false
return true
```

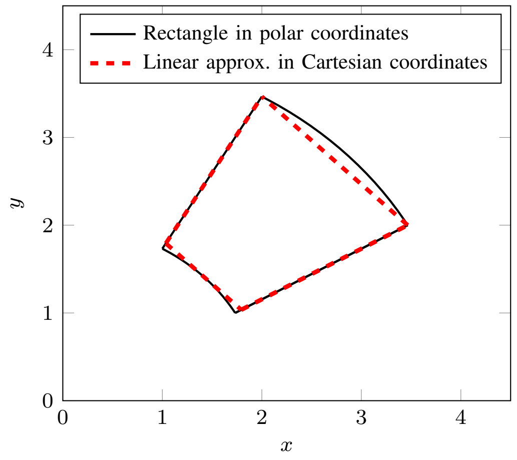


Fig. 7. Cartesian linear approximation of polar coordinates.

Implementation details: We adapted Marabou[6], a state-of-the-art DNN verification tool, to compute the robust volume ratio of a hypercube. We also used its parallelization capabilities to improve the solving time [18].

When computing the robust volume ratio, we kept partitioning an unrobust hypercube until the range along each of its dimensions was 5% of the full range of that dimension (as defined by the training data). Hypercube partitions were created by bisecting the widest intervals of the hypercube. The K parameter (the number of partitions to be created) in Algorithm 3 was set to be 4.

Results: For each network, Step 1 and Step 2 finished within 12 hours. The rows in Table III show the maximum, minimum, median and mean numbers of clusters and hypercubes. The training set for each network contains more than 810,000 data points; thus, our hypercube approach reduced the number of solver calls by an order of magnitude.

We randomly sampled a total of 36375 hypercubes. For each of these hypercubes, we computed the robust volume ratio. Running Marabou with 4 threads and a time limit of 20 minutes (wall clock time) was sufficient for most (36277) of the hypercubes. The median (mean) solving time for these queries was 3 seconds (14 seconds). Another 95 hypercubes required Marabou with 8 threads and a 2 hour (wall clock time) time limit. The last 3 hypercubes finished within 45 minutes using Marabou with 96 threads. The last set of hypercubes jobs were run on a cluster equipped with Intel Xeon E5-2699 v4 CPUs running CentOS 7.7.

The median robust percentage of the 45 tested networks is 99.66%, and the mean is 97.68%. 41 out of the 45 tested

TABLE III
STATISTICS OF CLUSTERS AND HYPERCUBES

	Max	Min	Median	Mean
# Clusters	7252	196	4971	5067
# Hypercubes	87631	2445	70834	75801

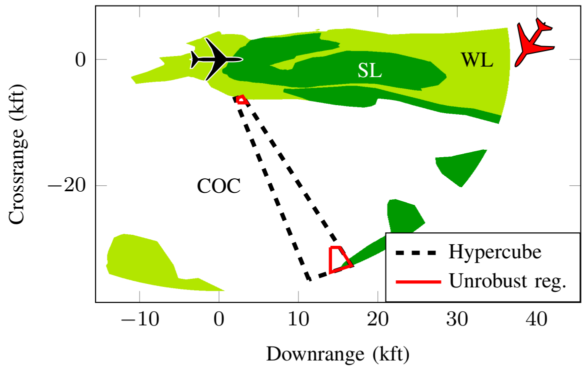


Fig. 8. Unrobust regions found for a COC region with $\tau = 40\,\text{s}$ & $s_{\text{adv}} = \text{SL}$.

networks have a robust percentage greater than 95%. For the remaining 4 networks, 3 have a robust percentage above 80%, while one has a robustness percentage of 67.03%.

Figure 8 shows one of the largest hypercubes created by the clustering algorithm. There are a few small regions where the expected advisory, COC, is not given, and our approach refines the large hypercube into smaller search regions in order to pinpoint all unrobust regions of the original hypercube.

Our robustness verification of the sampled hypercubes suggests that the trained networks in general have high local adversarial robustness. However, adversarial examples can be found for each network, and one of the 45 networks has relatively low robustness percentage. Therefore, robustness verification alone cannot guarantee full runtime safety of the networks. In the next section, we present an alternative approach that establishes confidence in the networks despite this.

V. Closed-Loop Verification of ACAS sXu

This section describes the dynamics of the ACAS sXu closed-loop system and then gives an overview of the reachability-based verification method. The dynamics and the reachability method are explained in detail in earlier work [7]. Later in the section, we present the verification results of our experiments.

To analyze the closed-loop system, a dynamical model must be assumed, which specifies how the state variables of the system change in response to the neural network advisories. The dynamics are a function of the ownship and intruder turn rates u_{own} and u_{int} respectively, and the advisory specifies the

TABLE IV
TURN RATE MODEL FOR CLOSED-LOOP SYSTEM

Aircraft	Advisory	$u_{\min}$ (°/s)	$u_{\max}$ (°/s)
Ownship	COC	$-\delta$	δ
Ownship	WL	$1.5-\delta$	$1.5+\delta$
Ownship	WR	$-1.5-\delta$	$-1.5+\delta$
Ownship	SR	$3.5-\delta$	$3.5+\delta$
Ownship	SL	$-3.5-\delta$	$-3.5+\delta$
Intruder	N/A	$-\delta$	δ

limits on these turn rates, as shown in Table IV. The δ term, assumed to be non-negative, is used to relax these limits, which allows us to show how the results of reachability analysis change for different dynamical models. This turn rate model is independent of how the horizontal table was formulated, and assuming a bounded range of turn rates rather than a distribution is needed for the reachability analysis.

The state variables are updated using the turn rate variables. If the state variables were represented using polar coordinates, the update equations would require functions like $\arctan$ and norm, which are difficult to handle with the reachability analysis and can lead to large approximation errors. These functions can be avoided by using Cartesian coordinates instead. The state variables are always defined with respect to the ownship's heading direction, which means that the ownship turn rate u_{own} will affect the updated intruder position as well as relative intruder heading angle, ψ.

Both aircraft are assumed to maintain constant turn rates u_{own} and u_{int}, respectively, for one second, which is the time between advisories. The position of the two aircraft after maneuvering as defined in the original coordinate frame centered at the ownship's original position is:

$$x'_{\text{own}} = v_{\text{own}} \frac{\sin(u_{\text{own}})}{u_{\text{own}}} \tag{3}$$

$$y'_{\text{own}} = v_{\text{own}} \frac{1 - \cos(u_{\text{own}})}{u_{\text{own}}} \tag{4}$$

$$x'_{\text{int}} = x + v_{\text{int}} \frac{\sin(\psi + u_{\text{int}}) - \sin(\psi)}{u_{\text{int}}} \tag{5}$$

$$y'_{\text{int}} = y + v_{\text{int}} \frac{\cos(\psi) - \cos(\psi + u_{\text{int}})}{u_{\text{int}}}. \tag{6}$$

Once the new positions of the two aircraft are computed in the original coordinate frame, the new state variables can be computed as the position of the intruder aircraft relative to the ownship's new position and heading direction using:

$$\begin{bmatrix} x \\ y \\ \psi \\ v_{\text{own}} \\ v_{\text{int}} \\ \tau \\ s_{\text{adv}} \end{bmatrix} \leftarrow \begin{bmatrix} (x'_{\text{int}} - x'_{\text{own}})\cos(u_{\text{own}}) + (y'_{\text{int}} - y'_{\text{own}})\sin(u_{\text{own}}) \\ (y'_{\text{int}} - y'_{\text{own}})\cos(u_{\text{own}}) - (x'_{\text{int}} - x'_{\text{own}})\sin(u_{\text{own}}) \\ \psi + u_{\text{int}} - u_{\text{own}} \\ v_{\text{own}} \\ v_{\text{int}} \\ \max(0, \tau - 1) \\ s'_{\text{adv}} \end{bmatrix} \tag{7}$$

As mentioned earlier, we have 50 different neural networks, one for each combination of s_{adv} and $\tau \in \{0, 1, 5, 10, 20, 40, 60, 80, 100, 101\}$. For the reachability analysis, each neural network has only three inputs, ρ, θ, and ψ, because the speed dimensions are assumed to be constant. When the neural networks are used, the τ value is rounded down to the closest value in $\tau \in \{0, 1, 5, 10, 20, 40, 60, 80, 100, 101\}$ and used with s_{adv} to determine the network to be evaluated.

This model assumes a constant speed for both ownship and intruder aircraft, though future work could incorporate speed changes into the model. The results generated in this section assume $v_{\text{own}} = 186\,\text{ft/s}$ and $v_{\text{int}} = 142\,\text{ft/s}$, which are discrete values used in the table. The same approach could be applied to different speed combinations to produce additional results.

A. Reachability Analysis

We now give a high-level overview of the reachability method proposed by [7]. The method uses open-source DNN verification tools with the system dynamics to determine which regions of the state space can be reached over time. DNN verification tools like Marabou [6] and Reluval [9] can determine whether an advisory is given at any point within a region of the input space. By splitting the input region into small cells and computing which advisories can be given within each cell, the method computes an over-approximation of the neural network system.

The analysis begins by partitioning the entire state space into a set $\mathcal{C}$ of cells and initializing a set of reachable cells, $\mathcal{R}_0$, which is the set of states that could occur before the neural network takes action. For collision avoidance networks, this would include all cells where either vertical or horizontal separation is at maximum sensing range. Then, for each t, we compute $\mathcal{R}_{t+1}$ as follows. For each c in R_t, we compute $\mathcal{A}_c$, the set of all possible advisories that the system can produce from points within c. The system dynamics are then used to compute $R_{c,a}$, the cells within $\mathcal{C}$ that could be reached at the next time step from some state within $c \in \mathcal{R}_t$ given advisory $a \in \mathcal{A}_c$. If the system dynamics are nonlinear, then $R_{c,a}$ is an over-approximation of the next states reachable from c. Then, $\mathcal{R}_{t+1}$ is computed as the union of all cells that intersect with $R_{c,a}\ \forall c \in \mathcal{R}_t,\ a \in \mathcal{A}_c$. This process is repeated until either an NMAC cell is added to the reachable set or $\mathcal{R}$ converges to a steady state with no NMAC cells.

The reachability analysis is summarized in Algorithm 4. Because of the over-approximation, if the reachability analysis concludes that the system cannot reach an unsafe state, then the real neural network system is guaranteed to maintain safety.

In general, the method may not converge to a steady-state set. In that case, the method should be repeated with a finer approximation of the system. This could be done either by using a finer grid or by refining over-approximations of nonlinear dynamics.

Algorithm 4 Reachability analysis [7] for collision avoidance neural networks

```
Input: $\mathcal{R}_0$, $\mathcal{C}$
  $t = 0$
  while isSafe($c$) $\forall c \in \mathcal{R}_t$ and ($t = 0$ or $\mathcal{R}_t \neq \mathcal{R}_{t-1}$)
    $t = t + 1$
    $\mathcal{R}_t \leftarrow \emptyset$
    for $c \in \mathcal{R}_{t-1}$
      Compute $\mathcal{A}_c$ using neural network verification tool
      for $a \in \mathcal{A}_c$
        Compute $R_{c,a}$ using state dynamics
        $\mathcal{R}_t \leftarrow \mathcal{R}_t \bigcup \{R_{c,a} \bigcap \mathcal{C}\}$
  if isSafe($c$) $\forall c \in \mathcal{R}_t$
    return Safe
  return Unsafe
```

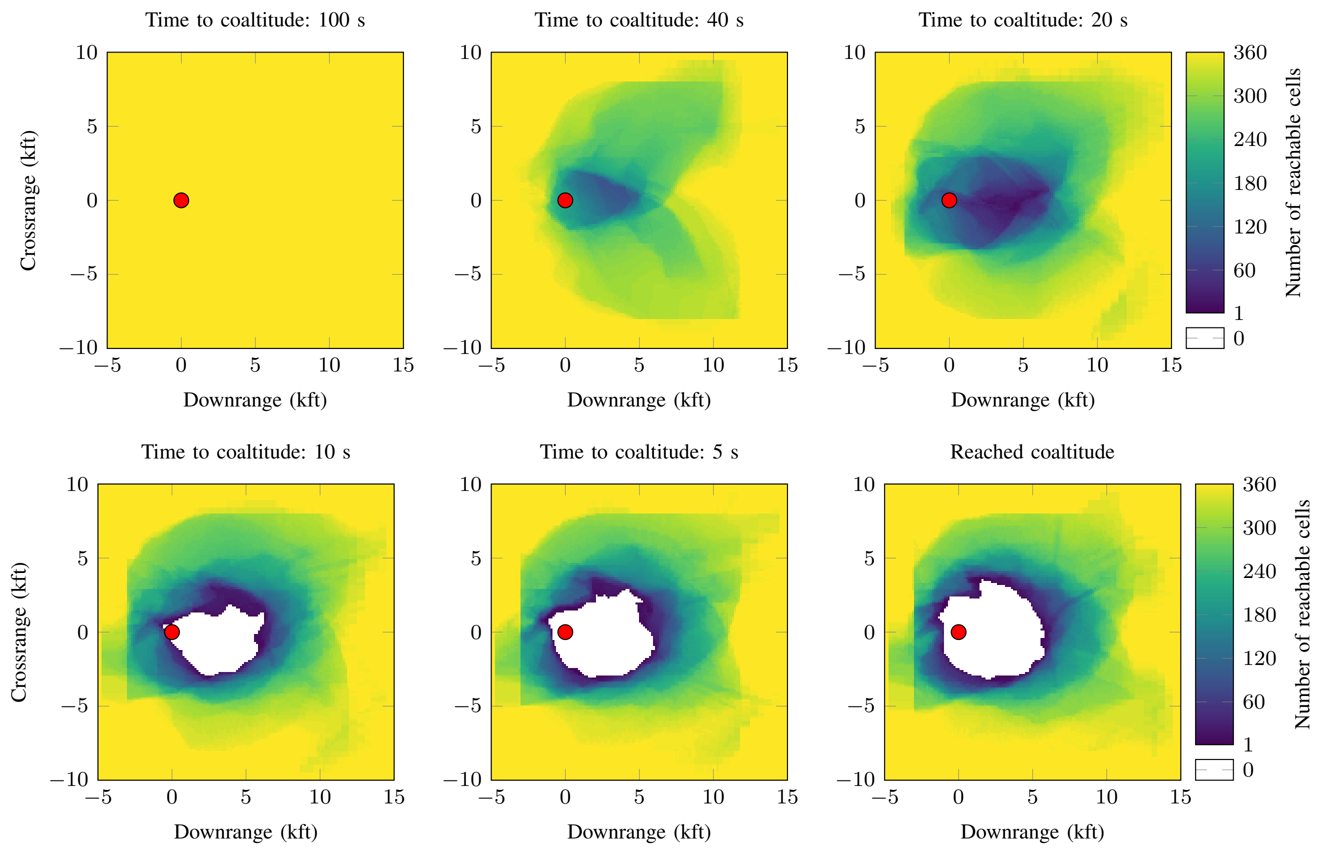


Fig. 9. Fine Grid – Reachable set for the networks over time.

B. *Experimental Evaluation*

The reachability method requires defining the search space grid, including the size of the grid and the granularity of the cells in the grid. We performed two experiments on the ACAS sXu closed-loop system, based on the cell granularity: first with a coarse grid, and later with a fine grid. For the grid size, we took the entire range of the input variables from the training data. We imposed a 16GB memory limit for the experiments.

Implementation Details: We adapted the reachability code developed previously [7]. The code required that the inputs be given in rectangular coordinates. Rectangular coordinates improve the reachability analysis. For the underlying DNN verification tool, we used ReluVal because of its use of symbolic bound propagation [9], which makes it especially fast for small regions of input space. For the dynamical model, we used $\delta = 0\,°/\mathrm{s}$, which assumes that the intruder does not turn, and that the ownship turns at precise turn rates. An intruder could appear far away horizontally or vertically, so the initial set is all states when $\tau = 100\,\mathrm{s}$, and cells that are at the maximum horizontal range are always added to the reachable set as τ counts down towards $0\,\mathrm{s}$, when vertical separation is lost.

Results using a Coarse Grid: For the coarse grid, the cells were discretized in x (downrange – 136 units) and y (crossrange – 140 units) more densely near the NMAC region

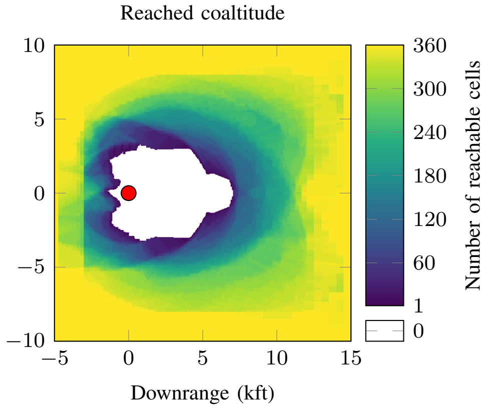


Fig. 10. Reachable set at $\tau = 0\,\mathrm{s}$ using the original table.

and areas leading to that region. In addition, ψ was discretized to 360 one-degree segments. The final discretization had 6.86 million cells. We ran Reluval on each network in parallel on a cluster. On average, each job required about 3 hours. The result of reachability analysis was not conclusive because of the over-approximation in the computation of the reachable set. This was expected due to the use of a coarse grid.

Results using a Fine Grid: We refined the coarse grid by more finely discretizing the cells in x (334 units) and y

(288 units). Similar to the coarse grid, we discretized more finely near the NMAC region and areas leading to that region. The discretization of ψ was unchanged. This resulted in 34.6 million cells in total. It required about 4 hours (on average) to run the Reluval job for each network. This time, we were able to show that no NMAC cells are reachable. Fig. 9 shows snapshots of the reachable cells at different τ values. The color of the cell in the plots indicates the number of cells reachable for that (x, y) location, i.e. the number of intruder heading angles reachable at that location. Yellow indicates more cells are reachable while blue shows fewer cells are reachable, and white means no cells are reachable. The red dot at the origin represents the NMAC region. Initially, all cells are added to the reachable set at $\tau = 100\,\text{s}$, but over time the neural network collision avoidance system removes unreachable cells around the NMAC region. Because the NMAC region is completely inside the white unreachable region when $\tau = 0\,\text{s}$, no NMACs are possible. The reachability method proves separation of $1000\,\text{ft}$, which is the minimum distance the intruder aircraft could be horizontally when the vertical separation between the ownship and the intruder is lost.

Comparison to Table Policy: The original ACAS sXu table used to train the neural networks can also be evaluated with the reachability analysis. Instead of using ReluVal to determine which advisories can be given within a cell, advisories are determined by the table values with a nearest-neighbor approximation. The rest of the reachability analysis follows the same procedure as with the neural networks.

Using the fine discretization, with $\delta = 0\,^\circ/\text{s}$, the reachability analysis could guarantee horizontal separation of only $781\,\text{ft}$ using the table, which is smaller than that of the neural network representation. Fig. 10 shows the reachable set once vertical separation is lost. This result suggests that the neural network representation does not degrade safety.

Results using Larger δ Values: The previous results use $\delta = 0\,^\circ/\text{s}$ which is restrictive and not representative of real-world systems with errors. Increasing delta expands the turn rate limits, which gives the aircraft more maneuverability but degrades the safety guarantees. For $\delta = 0.1\,^\circ/\text{s}$, the reachability analysis proves that the neural network will have $818.9\,\text{ft}$ of horizontal separation, while the table shows only $410\,\text{ft}$, which is less than the required $500\,\text{ft}$ to avoid an NMAC. If δ is increased to $0.2\,^\circ/\text{s}$, the horizontal separations at $\tau = 0\,\text{s}$ are $475\,\text{ft}$ and $279.5\,\text{ft}$ for the neural network and the table, respectively. Therefore, neither the neural network nor table representations could be proven safe with $\delta = 0.2\,^\circ/\text{s}$. However, this result does not mean that the table or neural network systems are unsafe because the reachable NMACs could be caused by over-approximation errors.

Results with Horizontal Separation Initial Set: The reachability analysis experiments described above model intruders that begin far away horizontally or vertically. The same reachability analysis can be used with different initial sets to study different types of situations. For example, coaltitude encounters can be simulated by setting the inital set to include all cells at the maximum horizontal separation when vertical separation is already lost ($\tau = 0\,\text{s}$). This initial set is depicted in the left plot of Fig. 11.

The reachability analysis was repeated for both the neural network and table representations using the coaltitude initial set. The results with $\delta = 0.0$ are depicted in Fig. 11, which show that both the neural network and table collision avoidance systems are able to maintain over 2500 ft of horizontal separation from the intruder aircraft. Table V aggregates the results of reachability analysis for both the neural network and table representations. Safety margins decrease as δ increases, and the neural network results have larger safety margins than those of the table for all $\delta > 0$. Using the coaltitude initial set allowed safety to be proven for larger δ values, suggesting that an intruder beginning above or below the ownship can cause problems for the reachability method. This could be addressed by reducing over-approximation errors by using a more finer grid. Overall, this reachability analysis shows that the prototype neural network ACAS sXu can be proven safe for a variety of dynamical models, and the neural network representation can be proven safe whenever the original table can be proven safe.

VI. Conclusion

We have presented a methodology for formally verifying a DNN-based collision avoidance system for small unmanned aircraft. Hypercube clustering can be used to verify local robustness of the trained networks. This approach allows us to check multiple single-point local robustness queries as one query. The results of local robustness show that the neural networks are not locally robust everywhere. However, by using a reachability method proposed in earlier work, we have shown that the closed-loop system with the neural network cannot reach an unsafe state. In the reachability analysis, we have assumed that the velocities of the ownship and the intruder are constant. Moreover, the analysis is based on over-approximating nonlinear dynamics of the closed-loop system and discretizing the input search space. Using a coarse search grid, we could not prove safety with the reachability method. As a consequence, we used a finer grid in the reachability analysis, and that has allowed us to prove safety of the closed-loop system.

This work can be extended in several directions. For local robustness, improving the clustering algorithm with polytopes would be an interesting future direction to explore. For the reachability analysis, relaxing the assumption about velocities

TABLE V
Reachability analysis results with coaltitude initial set

δ ($^\circ$/s)	Min Table Separation (ft)	Min Network Separation (ft)
0.0	2756.1	2600.5
0.1	1788.9	2545.3
0.2	1777.8	2438.2
0.3	540.8	992.5
0.4	485.4	922.0
0.5	459.6	820.1

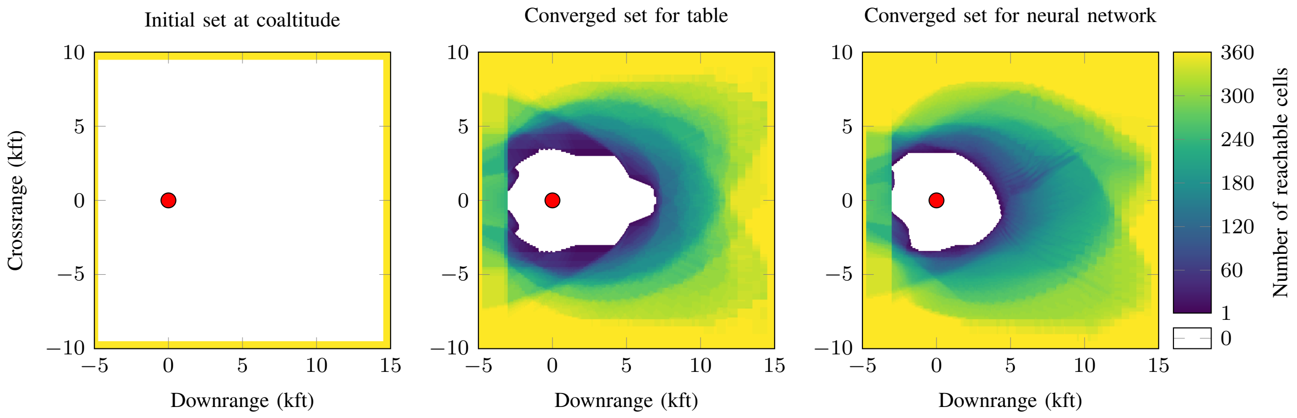


Fig. 11. Reachable sets using an initial set with maximum horizontal separation.

of the ownship and the intruder would be important. Another interesting direction would be automating the refinement of the over-approximation in the reachability method.

ACKNOWLEDGMENTS

This work was supported in part by the Stanford Center for AI Safety, as well as by the National Science Foundation under Graduate Research Fellowship DGE-1656518 and under grant number 1814369.

REFERENCES

[1] L. E. Alvarez, I. Jessen, M. P. Owen, J. Silbermann, and P. Wood, "ACAS sXu: Robust decentralized detect and avoid for small unmanned aircraft systems," in *Digital Avionics Systems Conference (DASC)*, 2019, pp. 1–9.

[2] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 2019.

[3] J. G. Lopez, L. Ren, B. Meng, R. Fisher, J. Markham, M. Figard, R. Evans, R. Spoelhof, M. Rubenstahl, and S. Edwards, "Integration and flight test of small UAS detect and avoid on a miniaturized avionics platform," in *Digital Avionics Systems Conference (DASC)*, 2019.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[5] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *International Conference on Learning Representations (ICLR)*, 2017.

[6] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The Marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer-Aided Verification*, 2019, pp. 443–452.

[7] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," in *Digital Avionics Systems Conference (DASC)*, 2019, pp. 1–10.

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

[9] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 6369–6379.

[10] K. A. Smith, A. E. Vela, M. J. Kochenderfer, and W. A. Olson, "Optimizing a collision-avoidance system for closely spaced parallel operations," *Journal of Aerospace Information Systems*, vol. 12, no. 10, pp. 618–633, 2015.

[11] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[14] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning (ICML)*, vol. 30, 2013, p. 3.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[16] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer-Aided Verification*, ser. Lecture Notes in Computer Science, vol. 10426, Springer, 2017, pp. 3–29.

[17] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 2613–2621.

[18] H. Wu, A. Ozdemir, A. Zeljić, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, and C. Barrett, "Parallelization techniques for verifying neural networks," in *Formal Methods in Computer Aided Design (FMCAD)*, 2020.