Verifying Aircraft Collision Avoidance Neural Networks Through Linear Approximations of Safe Regions

Kyle D. Julian* and **Mykel J. Kochenderfer**
Stanford University, Stanford, CA 94305

Shivam Sharma* and **Jean-Baptiste Jeannin**
University of Michigan, Ann Arbor, MI 48109

Abstract

The next generation of aircraft collision avoidance systems frame the problem as a Markov decision process and use dynamic programming to optimize the alerting logic. The resulting system uses a large lookup table to determine advisories given to pilots, but these tables can grow very large. To enable the system to operate on limited hardware, prior work investigated compressing the table using a deep neural network. However, ensuring that the neural network reliably issues safe advisories is important for certification. This work defines linearized regions where each advisory can be safely provided, allowing Reluplex, a neural network verification tool, to check if unsafe advisories are ever issued. A notional collision avoidance policy is generated and used to train a neural network representation. The neural networks are checked for unsafe advisories, resulting in the discovery of thousands of unsafe counterexamples.

Introduction

Over the last decade, neural network representations have become popular in decision making systems for a variety of domains. Neural networks are state-of-the-art for image recognition systems (Simonyan and Zisserman 2015; He et al. 2016) and can learn to play games at super-human levels (Mnih et al. 2015; Silver et al. 2016). In these domains, a mistake by the neural network may have minor consequences; however, neural networks can also be used in safety-critical systems where a failure could be catastrophic. For example, neural networks have been used to steer autonomous cars given images (Bojarski et al. 2016) and guide unmanned aircraft to waypoints (Julian and Kochenderfer 2017). If a neural network steers an autonomous car off the road or directs an aircraft into an obstacle, the result could be be expensive or lead to loss of life. In order for neural networks to be used for such applications, confidence must be established in their safe operation.

In the last few years, new research has resulted in tools to verify safety properties of neural networks. One tool, Reluplex, uses a Satisfiability Modulo Theories solver and extends the simplex method for neural networks with rectified linear unit (ReLU) activation functions to determine whether any input in a specified input region produces outputs with a desired property (Katz et al. 2017). Another approach defines neural network verification as a reachability problem that can be solved using a mixed integer linear program formulation (Lomuscio and Maganti 2017). Furthermore, a tool known as AI2 uses an overapproximation of the neural network to quickly verify safety properties of neural networks (Gehr et al. 2018). These tools enable network properties to be rapidly verified, but more work is needed to develop properties that will ensure safe operation of neural network systems.

This work focuses on the verification of neural networks used for aircraft collision avoidance. We created a highly simplified aircraft collision avoidance policy that uses vertical maneuvers using value iteration (Egorov et al. 2017). This policy, which we call VerticalCAS, is loosely based on an early prototype of the next generation airborne collision avoidance system for commercial aircraft, ACAS Xa (Kochenderfer 2015). Although VerticalCAS is not the ACAS Xa system that will be flown on real aircraft, VerticalCAS serves as a simple and open-source collision avoidance policy that can be used in the development of desirable properties. These properties should also hold for other vertical collision avoidance systems. After generating a collision avoidance policy, a neural network is trained to represent the original discrete policy (Julian et al. 2016).

Previous work has developed equations to verify the safety of the tabular collision avoidance policy by defining "safeable" regions for each advisory (Jeannin et al. 2017). In order to verify these properties for the neural network representation, the equations are linearized to enable the use of linear program solvers used by Reluplex. This paper describes the linearization process, introduces a new variable τ, the time to loss of horizontal separation, and describes the formulation and verification of "safeable" regions for neural networks.

VerticalCAS

The VerticalCAS collision avoidance system used throughout this paper is inspired by ACAS Xa, which frames aircraft collision avoidance as a Markov decision process (MDP) (Kochenderfer 2015). The ACAS Xa system is the successor to the current Traffic alert and Collision Avoidance System (TCAS) and provides pilots with advisories to

*Equal Contribution

change their vertical rate to prevent a possible near mid-air collision (NMAC). A NMAC is defined as an *intruder* aircraft coming inside the ownship *puck* which is described in Fig. 1 as a region $h_p = 100\,\text{ft}$ above and below, and $r_p = 500\,\text{ft}$ radially around the ownship aircraft (the aircraft where the collision avoidance system is installed).

VerticalCAS has 5 inputs which describe the system's state:

1. h (ft): Altitude of intruder relative to ownship
2. v_O (ft/s): ownship vertical climb rate
3. v_I (ft/s): intruder vertical climb rate
4. a_{prev}: previous advisory
5. τ (sec): time to loss of horizontal separation

The first 3 inputs are spatial and velocity quantities that are described in Fig. 1. Relative altitude h varies from −8000 ft to 8000 ft, and the aircraft climb rates vary from −100 ft/s to 100 ft/s.

Previous advisory (a_{prev}) dictates which advisories VerticalCAS can issue given the most recent advisory. This restricts the network from issuing conflicting advisories like strong ascend or descend advisories immediately after a clear of conflict advisory which can be confusing to pilots.

Time to loss of horizontal separation (τ) is the time till the horizontal separation between the intruder and ownship is less than r_p. A more explicit definition of τ is

$$\tau = \frac{r - r_p}{r_v} \tag{1}$$

where r is the horizontal separation between the ownship and intruder, and r_v is the relative horizontal velocity between the two aircraft.

Markov Decision Process Policy

VerticalCAS is computed using local approximation value iteration as implemented by the Julia package called POMDPs.jl (Egorov et al. 2017). The states, dynamics, rewards, and advisories reflect an early prototype of the ACAS Xa system described by Kochenderfer (2015). Each state $s \in \mathcal{S}$ represents a discrete encounter geometry between the ownship and intruder aircraft and has five dimensions which are the inputs outlined above.

The system issues a new advisory a every ϵ seconds, and there are nine possible advisories as described in Table 1, where g is Earth's sea-level gravitational acceleration (Jeannin et al. 2017). Each vertical advisory is defined by a target velocity v_{lo} and sign w. If $w = 1$, the ownship can assume a velocity in the range $[v_{lo}, +\infty)$, and if $w = -1$ the ownship can assume a velocity in the range $(-\infty, v_{lo}]$. In addition to the advisory (w, v_{lo}), the ownship has to accelerate at least a_{lo} until it is in the acceptable velocity range defined by the issued advisory.

The transition model $T(s, a, s')$ and reward model $r(s, a)$ used for vertical collision avoidance are explained in previous work (Kochenderfer 2015). Local approximation value iteration is used to compute the state-action values, $Q(s, a)$,

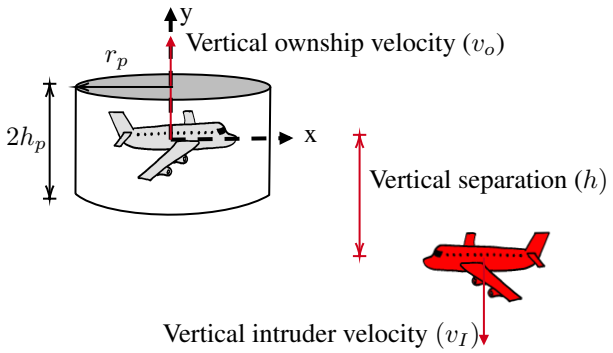

Figure 1: Three input variables of VerticalCAS neural network and ownship puck defined by r_p and h_p on ownship centered coordinate frame.

such that the finite-horizon Bellman equation holds for all states and actions:

$$Q(s, a) = r(s, a) + \sum_{s'} \max_{a'} T(s, a, s') Q(s', a') \tag{2}$$

Because s' might not be exactly one of the discrete states $s \in S$, multilinear interpolation is used to compute $Q(s', a')$. After computing the Q values using local approximation value iteration, the advisory associated with the highest Q value for a given state is the best advisory and is issued by the system. In addition, because the computed policy tends to advise Clear of Conflict in cases where an NMAC is imminent and unavoidable, the advisory at time $\tau = 6$ is used in situations where $\tau < 6$.

Neural Network Representation

Storing the MDP policy with fine resolution in a table format can require large amounts of storage space, which may prevent implementation on limited avionics hardware. One approach to compressing the policy representation approximates the policy using a neural network through the use of supervised learning and an asymmetric loss function, which encourages the neural network to simultaneously approximate the Q-values and highest scoring advisory (Julian et al. 2016). One network was trained for each previous advisory a_{prev}, resulting in nine fully connected neural networks using six hidden layers of 45 hidden units each. Each hidden layer uses rectified linear unit (ReLU) activation, which is defined as $\text{ReLU}(x) = \max(0, x)$ (Dahl, Sainath, and Hinton 2013). Each network uses the remaining four state variables as inputs and outputs a value associated with each possible advisory. Each neural network was trained for 200 epochs using AdaMax optimization (Kingma and Ba 2015) implemented in Keras (Chollet 2015) with the Theano backend (Theano Development Team 2016), which requires an hour to train on an NVidia Titan X GPU.

Figure 2 plots the advisory the system would give to the ownship if the intruder were at each location in the plot. In this scenario, the ownship is climbing while the intruder is maintaining a constant altitude. If the intruder is approaching the ownship from above, the system alerts the ownship to

Table 1: VerticalCAS advisories

Advisory	**Description**	**Vertical Range** (Min, Max) [ft/min]	**Strength** a_{lo}	**Sign** w	**Advisory** v_{lo} [ft/min]
COC	Clear of Conflict	$(-\infty, +\infty)$	$g/4$	N/A	N/A
DNC	Do Not Climb	$(-\infty, 0]$	$g/4$	-1	0
DND	Do Not Descend	$[0, +\infty)$	$g/4$	$+1$	0
DES1500	Descend at least 1500 ft/min	$(-\infty, -1500]$	$g/4$	-1	-1500
CL1500	Climb at least 1500 ft/min	$[+1500, +\infty)$	$g/4$	$+1$	$+1500$
SDES1500	Strengthen Descend to at least 1500 ft/min	$(-\infty, -1500]$	$g/3$	-1	-1500
SCL1500	Strengthen Climb to at least 1500 ft/min	$[+1500, +\infty)$	$g/3$	$+1$	$+1500$
SDES2500	Strengthen Descend to at least 2500 ft/min	$(-\infty, -2500]$	$g/3$	-1	-2500
SCL2500	Strengthen Climb to at least 2500 ft/min	$[+2500, +\infty)$	$g/3$	$+1$	$+2500$

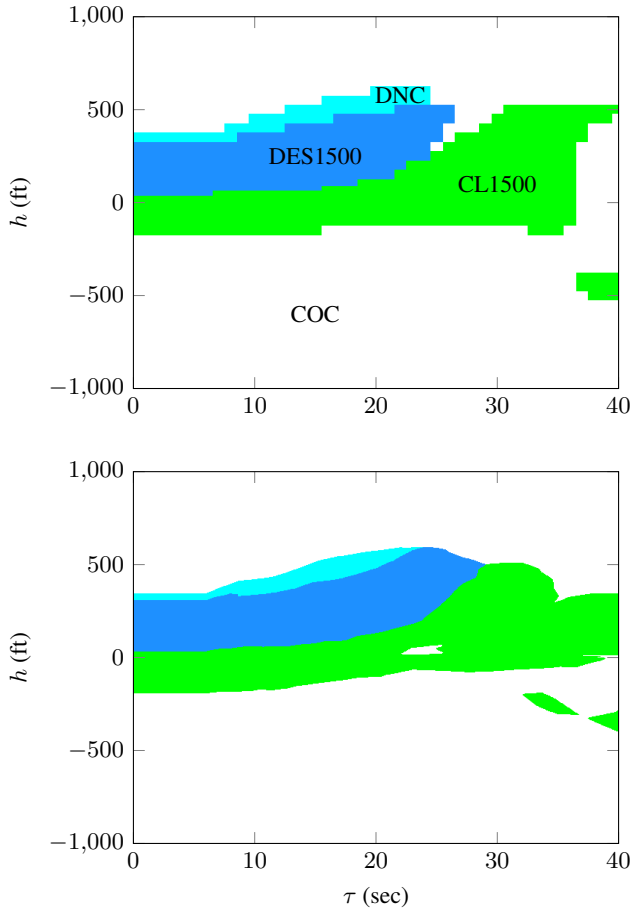

Figure 2: Example policy plots for a climbing ownship and level-flying intruder using the MDP table (top) and neural network (bottom)

stop climbing (DNC) or descend (DES1500) in order to prevent an NMAC. If the intruder is a little below the ownship, the system advises the pilot to continue climbing (CL1500). In other locations, a collision is not imminent and the system alerts clear-of-conflict (COC). The neural network representation is a smooth approximation of the original table policy. Although the network appears to represent the table well, verification is needed to ensure that the neural network alerts safely at all times.

Safe Regions

The *safe region* is defined as the region in space where an intruder aircraft will be safe (i.e. will not enter the ownship puck), given the ownship aircraft is following a single advisory (shown in Fig. 3). The safe region is described by the ownship travelling along a nominal trajectory. This nominal trajectory is described by the ownship following an advisory exactly, i.e., if the advisory issued allows a range of velocities $[1500, +\infty)$, the nominal trajectory will be defined by the ownship assuming a velocity of 1500 ft/min. From our earlier definition of $\tau = \frac{r - r_p}{r_v}$, $\tau = 0$ of the nominal trajectory is at $r = r_p$.

The nominal trajectory of the ownship is simply a parabolic trajectory due to constant vertical acceleration. The trajectory can be written as follows (Jeannin et al. 2017):

$$h_n = \begin{cases} \frac{a_{lo}}{2}\tau^2 + v_O\tau, & \text{if } 0 \le \tau < \frac{v_{lo} - v_O}{a_{lo}} \\ v_{lo}\tau - \frac{(v_{lo} - v_O)^2}{2a_{lo}}, & \text{if } \frac{v_{lo} - v_O}{a_{lo}} \le \tau \end{cases} \quad (3)$$

where h_n is the altitude of the ownship in a coordinate frame centered in the starting position of the ownship Fig. 1 (Note: the subscript n denotes the nominal trajectory). The piecewise Eq. (3) describes the dynamics when the ownship velocity is less than v_{lo} and when the ownship velocity is greater than v_{lo}. Once the ownship climb rate reaches v_{lo}, the aircraft is compliant with the advisory and continues climbing with no vertical acceleration.

For the example of CL1500, the safe region will be the region below the 'puck' of the ownship flying along the CL1500 nominal trajectory. If an intruder is in this region below the ownship, it will be safe from collision for an ownship following the CL1500 advisory. Therefore, this region is defined as the *safe region* for a particular advisory.

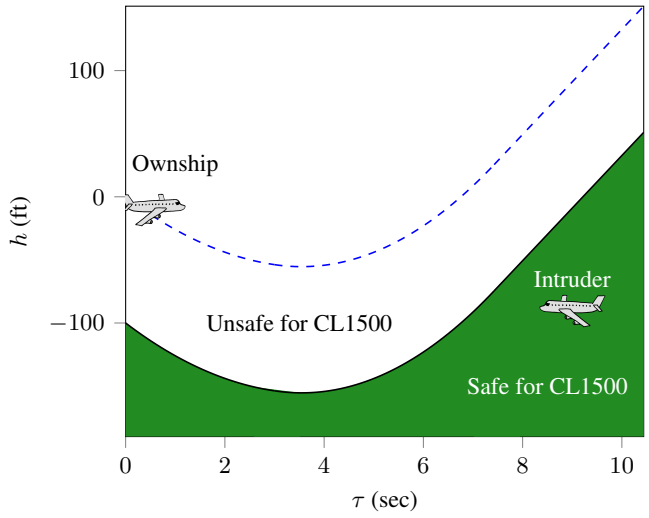

Figure 3: Intruder in the safe region for ownship advisory CL1500 and $v_O < 0$

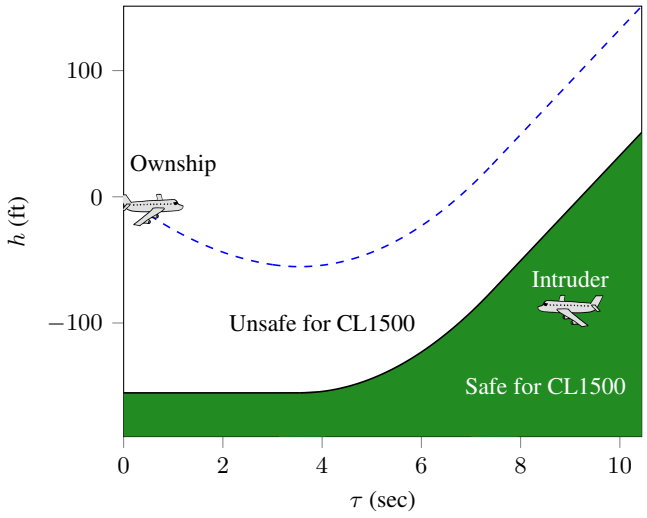

Figure 4: Intruder in the worst-case safe region for the ownship advisory CL1500 and $v_O < 0$

Safe regions have to be able to be represented in terms of network variables to define a search space in the state-space of the network. Representing safety bounds solely in terms of the five network variables poses some challenges which are discussed in the next section.

One limitation with using safe regions to verify safety properties is that safe regions assume that the ownship follows a single advisory throughout the encounter. In reality, multiple advisories can be issued during an encounter, giving the system an opportunity to change the advisory. Therefore, an advisory that was initially *unsafe* can be made *safe* with a change later in the encounter. Safeable regions, described below, build on the safe region concept and tackle this shortcoming.

Worst-Case Scenario Approach

An NMAC is defined as an intruder aircraft coming inside the ownship *puck*, as depicted in Fig. 1. The network is trying to prevent NMAC's, so the safe region is described by this puck around the ownship aircraft.

In Fig. 3, when the ownship is descending, the safe region bounds are described by the 'back' of the ownship puck (Jeannin et al. 2017), where the 'back' of the ownship puck can be represented as $\tau_{back} = \tau - \frac{2r_p}{r_v}$, where r_p is a known constant, but, r_v is unknown because it is not an input to the network. Thus, a worst-case approximation must be made to define the safe region bounds described by the back of the ownship puck.

At $\tau = 0$ the horizontal separation between the intruder and ownship is r_p. After this point, horizontal separation between the ownship puck and the intruder will not be regained again until $t = \frac{2r_p}{r_v}$ seconds in the future when the intruder will cross the back of the ownship puck. In the worst-case, $r_v \to 0$, and horizontal separation may never be regained. As a result, the intruder must be at an altitude that is safeable for all time $t > \tau$.

The relative horizontal velocity of the two aircraft r_v effectively dictates the width of the ownship puck in τ-space. The worst-case safe region bound should include all other unsafe regions, which is achieved as $r_v \to 0$ or the ownship puck is infinitely wide. The worst-case safe region bounds can be seen in Fig. 4. Using this approach, a worst-case safe region bound can be described where $\Omega_{\mathsf{Unsafe}} \subseteq \Omega_{\mathsf{Unsafe(worst\ case)}}$ i.e. all possible unsafe regions are subsets of the worst-case scenario unsafe region.

Safeable Regions

Safeable regions are defined as regions which are currently safe or that can be made safe in the future. A safeable region is constructed by assuming two worst-case trajectories of an aircraft complying with an advisory for time ϵ (VerticalCAS issues a new advisory every ϵ seconds). After ϵ, these two trajectories represent the two extreme positions of the ownship that complies with the initial advisory. From this point, the strongest reversing and strengthening advisories that VerticalCAS can issue are considered. If either of these advisories prevent a collision, then the intruder is in a safeable region. As a result, a collision with an intruder in the safeable region can always be avoided. For example, as seen in Figure 5, if the intruder is located as shown, the system can safely issue a CL1500 advisory because a strong reversal at the next time step will ensure that the ownship descends before reaching the intruder. A more detailed explanation of safeable regions is provided in (Jeannin et al. 2017).

If the system always gives safeable advisories whenever possible, then an intruder beginning in the safeable region will always be avoided. As a result, ensuring safety when

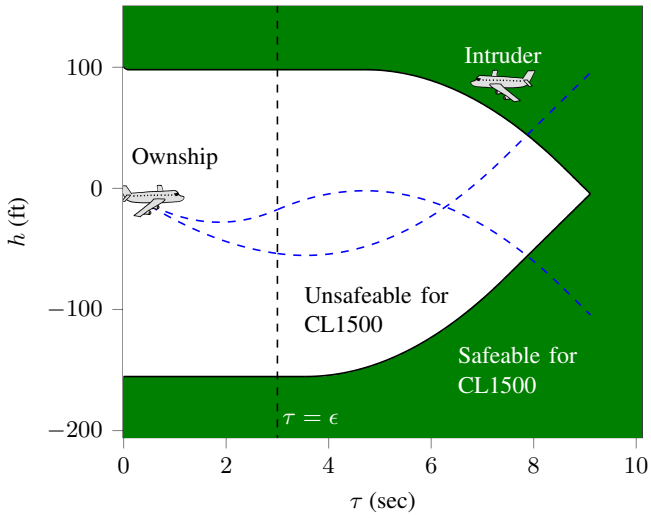

Figure 5: Safeable region with strengthen and reversal alerts issued at $\tau = \epsilon$.

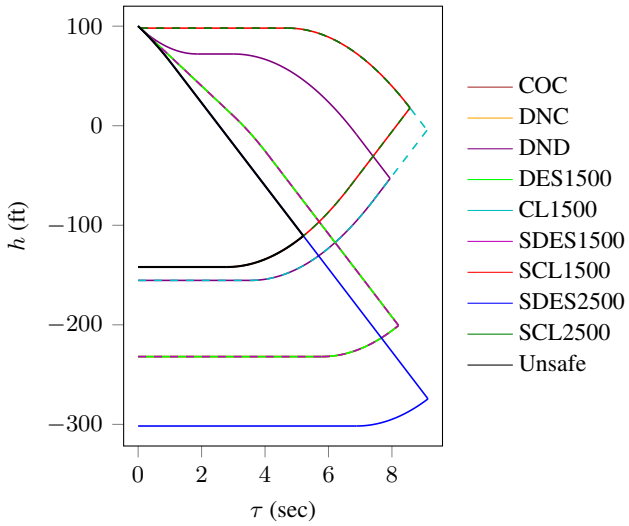

Figure 6: All unsafeable regions and the region that is unsafeable for all advisories

using the neural network system requires checking for any instances when the neural network gives an unsafeable advisory when a safeable advisory exists. To generate these regions, the region that is unsafeable for all advisories must be computed, which can be done by generating the intersection of all possible advisories, as illustrated in Fig. 6. The region to verify is shown in red in Fig. 7 because this region is unsafeable for CL1500 but would be safeable for another advisory such as SCL2500 or SDES2500. The next section describes how these safeable regions are adapted for use with the Reluplex neural network verification tool.

Checking Safeable with Reluplex

Reluplex extends the simplex method to verify neural network properties by representing neural networks, activation functions, and constraints as piecewise linear equations. Linear bounds are placed on the input variables to define the search region, and the output variables are constrained such that the advisory of interest must be associated with the largest valued output from the network. Reluplex systematically searches for an input to satisfy both input and output constraints. (Katz et al. 2017). The red unsafeable region in Fig. 7 is nonlinear and non-convex, so the region cannot be verified using Reluplex in the current form.

There are three adjustments made to the safeable regions to prepare the regions for Reluplex. First, the safeable regions are functions of the ownship's initial climbrate, which can vary from −100 ft/s to 100 ft/s. In order to avoid verifying every possible region generated by all floating point values of ownship climb rate, the regions are generated assuming a small range of climb rates instead of a single climb rate. To generate the safeable boundaries, the upper and lower trajectories are generated assuming the worst case initial climb rate. As a result, the unsafeable boundaries grow outwards, as seen in Fig. 8, which shows the safeable region boundaries for different ranges of climb rates.

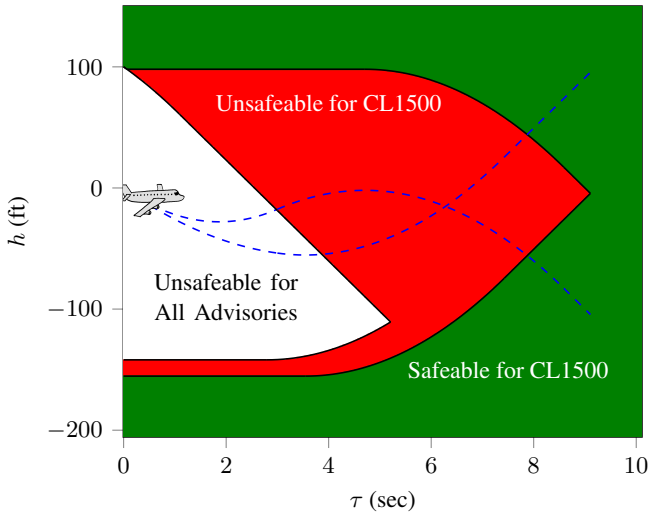

Figure 7: Safeable and unsafeable regions for CL1500 advisory

Next, the safeable regions are linearized so that boundaries can be represented in Reluplex. The linearization over-approximates the unsafeable region by approximating quadratic bounds as a piecewise linear function. The approximation uses either an inner approximation connecting points on the curve, or an outer approximation using line segments tangent to the curve. The type of approximation used is chosen to over-approximate the unsafeable region. Figure 9 shows the linearization of the safeable region that over-approximates the unsafeable region.

Lastly, the region checked by Reluplex is split into small

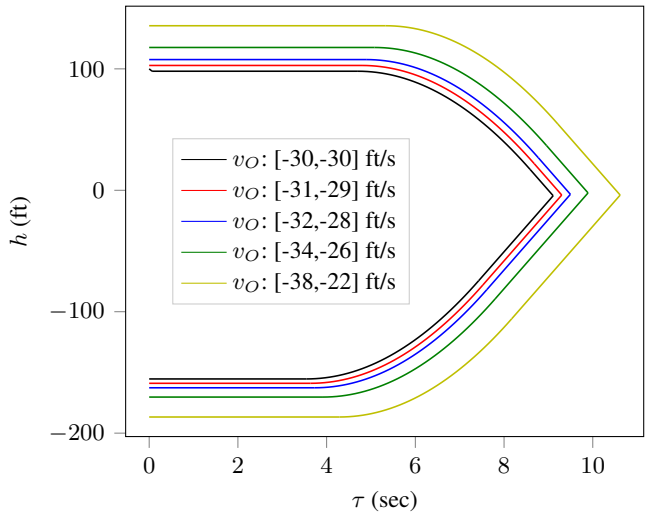

Figure 8: Safeable regions for different initial climbrates for the ownship

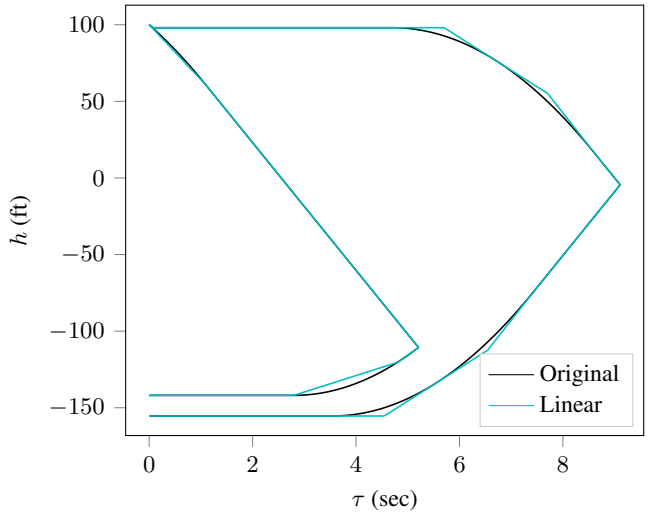

Figure 9: Over-approximation of the search region

slices that are defined by a lower and upper bound on τ as well as a single linear lower bound on h and a single linear upper bound on h. Because the neural network uses $\tau = 6$ for inputs where $\tau = 6$, the τ bounds are adjusted to ensure the network is evaluated at $\tau = 6$ for inputs where $\tau < 6$. Each small slice is checked as a separate query with Reluplex. A satisfiable set of inputs found by Reluplex represents a counterexample, or a set of network inputs that produce an unsafeable advisory when a safeable advisory exists. Because Reluplex is sound and complete, if Reluplex cannot find a counterexample for a query, then no counterexample exists.

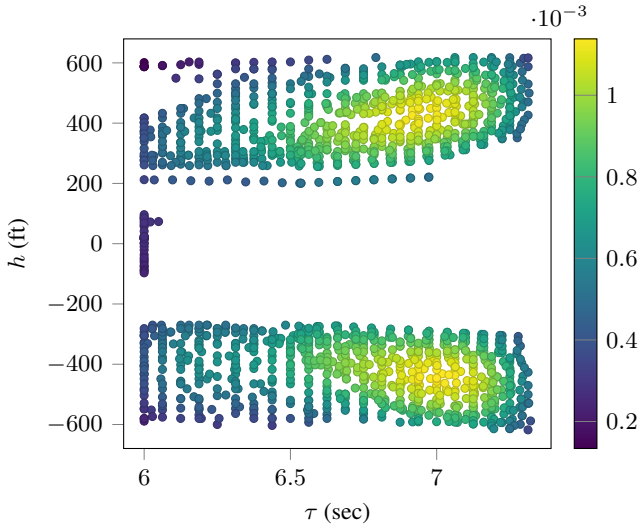

Figure 10: Heat map of counterexamples for a_{prev}: Clear of Conflict

Results

To verify the unsafeable regions in all of the neural networks, each of the nine neural networks associated with one of the previous advisories is evaluated for all allowed advisories. Using a Δv_O of 2 ft/s, there are 100 velocity ranges to verify. After slicing up each unsafeable region into small regions with linear bounds, a total of $42,032$ separate queries were generated and evaluated with Reluplex, which required 11 hours when using 9 independent threads. Each query was run with $\epsilon = 1$ second (in all the figures $\epsilon = 3$ seconds just for illustration purposes). As a result, $3,957$ counterexamples were discovered, about 9.14% of all queries. A table of when these counterexamples occurred is shown in Table 2, where N/A is used for advisories that are not allowed given the previous advisory. Most counterexamples occur for the COC advisory, but many other counterexamples exist for other advisories as well.

Visualizing the counterexamples in the form of a heat map allows for analysis of the network's performance. Fig. 10 plots all the counterexamples found by Reluplex for advisories issued after a clear of conflict advisory. No counterexamples are found in the white region in the middle of the plot because this region is unsafeable for all advisories and is omitted from the search region, as illustrated in Fig. 7. The lighter points represent a higher probability density of counterexamples. The figure illustrates that counterexamples are most prevalent at around $\tau = 7\,\text{s}$. This information can be useful for tweaking networks to perform safely. Also, Fig. 10 shows rough vertical stripes, which is due to the preference of Reluplex to return SAT points that occur along the boundary of a region rather than somewhere in the middle of a region.

In addition to the $42,032$ separate queries run that are summarized in Table 2, we ran $143,048$ queries on 25 in-

Table 2: Number of counterexamples discovered with Reluplex

	Current Advisory								
Previous Advisory	COC	DNC	DND	DES1500	CL1500	SDES1500	SCL1500	SDES2500	SCL2500
COC	359	28	0	48	21	N/A	N/A	N/A	N/A
DNC	438	30	0	40	47	N/A	N/A	N/A	N/A
DND	249	0	17	133	50	N/A	N/A	N/A	N/A
DES1500	284	0	1	1	0	65	76	N/A	N/A
CL1500	223	0	0	0	0	117	21	N/A	N/A
SDES1500	281	0	0	0	0	26	6	32	65
SCL1500	238	0	3	0	0	53	66	43	51
SDES2500	324	0	0	0	0	12	1	89	25
SCL2500	209	0	12	0	0	52	15	48	58

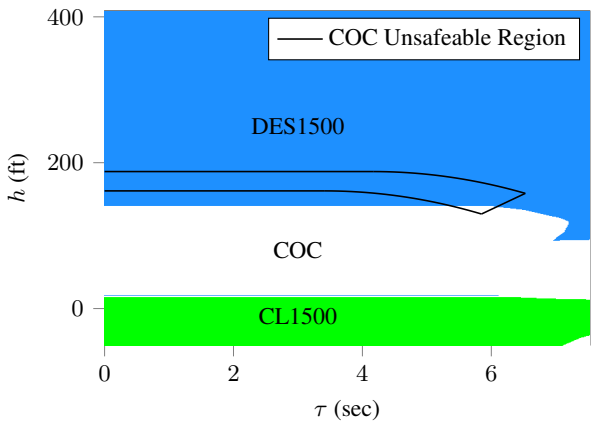

Figure 11: Unsafeable region for COC containing a counterexample

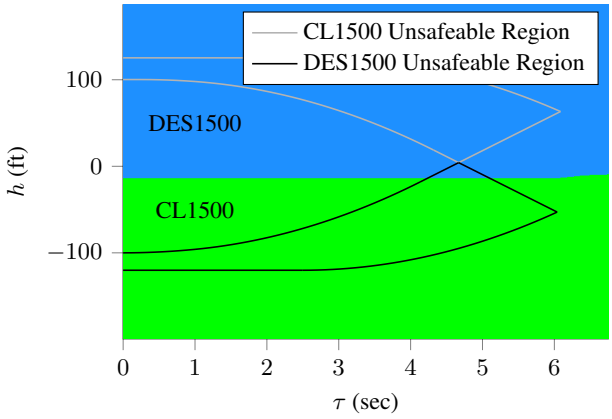

Figure 12: Unsafeable regions for CL1500 and DES1500 with DES1500 counterexample

dependent threads to study the effect of the linearization approximation on the number of counterexamples generated. All advisories were checked for a_{prev} = COC, linearized with line segment lengths of $\tau = 0.125, 0.25, 0.5, 1.0$, and 2.0 seconds for both under and over-approximation. All linear segments were split into small regions of the same size so that the number of regions generated remained the same for all cases. Neither the method of linearization nor the level of discretization had any affect on the number of counterexamples found. For each level of discretization $1,476$ counterexamples were found for both the under-approximation and over-approximation method. This is most likely due to the fact that counterexamples are usually found around linear parts of the safeable bounds, so finer linearization had no impact.

Some of these counterexamples are informative, and visualizing the policy at these points reveals problems that need to be addressed. For example, Fig. 11 shows the unsafeable region for COC, which extends into a large area of COC. Given that the unsafeable region appears at low τ and h, a collision is imminent, and COC is not safe to give. This information can be used to refine the policy and network to discourage COC advisories in these situations.

Many counterexamples are found at the boundary between two alerting regions. As shown in Figure 12, the regions being checked for DES1500 and CL1500 meet at a point. In order to avoid any counterexamples, the boundary between DES1500 and CL1500 must pass exactly through the point that divides the two unsafe regions. However, because the neural network is an approximation, the boundary is a little off, and a counterexample is discovered. In addition, no other advisory is safeable around the meeting point, so there is no other advisory the network could give to avoid a counterexample. Requiring the network to change advisories at an exact point in order to prove safety is too strict, so more work is needed to relax this requirement while still guaranteeing safety.

Conclusions and Future Work

After generating collision avoidance networks, linear safeable regions were defined for all possible advisories. The safeable regions define when an advisory can be made safe in the future, so that advisory is safe to give in the safeable region. If the system always gives safeable advisories when possible, then safety is guaranteed assuming the intruder begins in the safeable region. The safeable regions were checked with Reluplex, resulting in the discovery of

thousands of counterexamples. The counterexamples can be used to refine the neural networks to improve safety.

A primary issue with proving safety using safeable regions is the hard safety requirement imposed on neural networks. The safeable property requires that the boundary between advisories given by the neural network must pass through an exact point in the state space. In reality, no neural network will be able to satisfy such a hard requirement in all situations.

To overcome this challenge, we have been exploring an extension to safeable, which we call safeable2. A safeable2 region is defined as a region that is safeable by at least two advisories. Verifying safety with safeable2 removes the hard requirement of the neural network having to switch advisories at a single point, but rather allows a small region to switch advisories. In addition, safeable2 omits a small region of uncertain behavior (the region that is safeable by only a single advisory) around the unsafeable region where a lot of counterexamples are found. It will be interesting to explore the implications of using safeable2 to verify safety and whether this method eliminates spurious counterexamples to safe operation. Furthermore, future work will model pilot delay to ensure safety can be guaranteed with realistic pilot compliance.

References

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Chollet, F. 2015. Keras: Deep learning library for Theano and TensorFlow.

Dahl, G. E.; Sainath, T. N.; and Hinton, G. E. 2013. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 8609–8613. IEEE.

Egorov, M.; Sunberg, Z. N.; Balaban, E.; Wheeler, T. A.; Gupta, J. K.; and Kochenderfer, M. J. 2017. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research* 18(26):1–5.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (SP)*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Jeannin, J.-B.; Ghorbal, K.; Kouskoulas, Y.; Schmidt, A.; Gardner, R.; Mitsch, S.; and Platzer, A. 2017. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *International Journal on Software Tools for Technology Transfer* 19(6):717–741.

Julian, K. D., and Kochenderfer, M. J. 2017. Neural network guidance for UAVs. In *AIAA Guidance, Navigation, and Control Conference*, 1743.

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

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

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kochenderfer, M. J. 2015. *Decision Making Under Uncertainty: Theory and Application*. MIT Press.

Lomuscio, A., and Maganti, L. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.