NNLander-VeriF: A Neural Network Formal Verification Framework for Vision-Based Autonomous Aircraft Landing

Ulices Santa Cruz(✉) and Yasser Shoukry

University of California Irvine, Irvine, CA, USA
{usantacr,yshoukry}@uci.edu

Abstract. In this paper, we consider the problem of formally verifying a Neural Network (NN) based autonomous landing system. In such a system, a NN controller processes images from a camera to guide the aircraft while approaching the runway. A central challenge for the safety and liveness verification of vision-based closed-loop systems is the lack of mathematical models that captures the relation between the system states (e.g., position of the aircraft) and the images processed by the vision-based NN controller. Another challenge is the limited abilities of state-of-the-art NN model checkers. Such model checkers can reason only about simple input-output robustness properties of neural networks. This limitation creates a gap between the NN model checker abilities and the need to verify a closed-loop system while considering the aircraft dynamics, the perception components, and the NN controller. To this end, this paper presents NNLander-VeriF, a framework to verify vision-based NN controllers used for autonomous landing. NNLander-VeriF addresses the challenges above by exploiting geometric models of perspective cameras to obtain a mathematical model that captures the relation between the aircraft states and the inputs to the NN controller. By converting this model into a NN (with manually assigned weights) and composing it with the NN controller, one can capture the relation between aircraft states and control actions using one augmented NN. Such an augmented NN model leads to a natural encoding of the closed-loop verification into several NN robustness queries, which state-of-the-art NN model checkers can handle. Finally, we evaluate our framework to formally verify the properties of a trained NN and we show its efficiency.

Keywords: Neural network · Formal verification · Perception

1 Introduction

Machine learning models, like deep neural networks, are used heavily to process high-dimensional imaging data like LiDAR scanners and cameras. These data

This work was supported by the National Science Foundation under grant numbers #2002405 and #2013824.

J. V. Deshmukh et al. (Eds.): NFM 2022, LNCS 13260, pp. 213–230, 2022.
https://doi.org/10.1007/978-3-031-06773-0_11

driven models are then used to provide estimates for the surrounding environment which is then used to close the loop and control the rest of the system. Nevertheless, the use of such data-driven models in safety-critical systems raises several safety and reliability concerns. It is unsurprising the increasing attention given to the problem of formally verifying Neural Network (NN)-based systems.

The work in the literature of verifying NNs and NN-based systems can be classified into *component-level* and *system-level* verification. Representatives of the first class, namely *component-level* verification are the work on creating specialized decision procedures that can reason about input-output properties of NNs [2,8,10,11,17,18,20,25,26]. In all these works, the focus is to ensure that inputs of the NN that belong to a particular convex set will result in NN outputs that belong to a defined set of outputs. Such input-output specification allows designers to verify interesting properties of NN like robustness to adversarial inputs and verify the safety of collision avoidance protocols. For a comparison between the details and performance of these NN model checkers, the reader is referred to the annual competition on verification of neural networks [1]. Regardless of the improvements observed every year in the literature of NN model checkers, verifying properties of perception and vision-based systems as a simple input-output property of NNs is still an open challenge.

On the other hand, *system-level* verification refers to the ability of reasoning about the temporal evolution of the whole system (including the NNs) while providing safety and liveness assurance [7,12,23,24]. A central challenge to verify systems that rely on vision-based systems and other high-bandwidth signals (e.g., LiDARs) is the need to explicitly model the imaging process, i.e., the relation between the system state and the images created by cameras and LiDARs [23]. While first steps were taken to provide formal models for LiDAR based systems [23], very little attention is given to perception and vision-based systems. In particular, current state-of-the-art aims to avoid modeling the perception system formally, and instead focus on the use of abstractions of the perception system [14,19]. Unfortunately, these abstractions are only tested on a set of samples and lack any formal guarantees in their ability to model the perception system formally. Other techniques use the formal specifications to guide the generation of test scenarios to increase the chances of finding a counterexample but without the ability to formally prove the correctness of the vision-based system [12].

Motivated by the lack of formal guarantees of the abstractions of perception components [14,19], we argue in this paper for the need to formally model such perception components. Fortunately, such models were historically investigated in the literature of machine vision before the explosion of using data-driven approaches in machine learning [9,21]. While these physical/geometrical models of perception were shown to be complex to design vision-based systems with high performance, we argue that these models can be used for verification. In other words, we employ the philosophy of data-driven design of vision-based systems and model-based verification of such systems.

In this paper, we employ our philosophy above to the problem of designing a vision-based NN that controls aircraft while approaching runways to per-

form autonomous landing. Such a problem enjoys geometric nature that can be exploited to develop a geometrical/physical model of the perception system, yet represent an important real-world problem of interest to the autonomous systems designers. In particular, we present NNLander-VeriF, a framework for formal verification of vision-based autonomous aircraft landing. This framework provides several contributions to the state of the art:

- The proposed framework exploits the geometry of the autonomous landing problem to construct a formal model for the image formation process (a map between the aircraft states and the image produced by the camera). This formal model is designed such that it can be encoded as a neural network (with manually chosen weights) that we refer to as the perception NN. By augmenting the perception NN along with the NN controller (which maps camera images into control actions), we obtain a formal relation between the aircraft states and the control action that is amenable to verification.
- The proposed framework uses symbolic abstraction of the physical dynamics of the aircraft to divide the problem of model checking the system-level safety and liveness properties into a set of NN robustness queries (applied to the augmented NN obtained above). Such robustness queries can be carried out efficiently using state-of-the-art component-level NN model checkers.
- We evaluated the proposed framework on a NN controller trained using imitation learning.

2 Problem Formulation

Notation. We will denote by $\mathbb{N}$, $\mathbb{B}$, $\mathbb{R}$ and $\mathbb{R}^+$ the set of natural, Boolean, real, and non-negative real numbers, respectively. We use $||x||_\infty$ to denote the infinity norm of a vector $x \in \mathbb{R}^n$. Finally, we denote by $\mathcal{B}_r(c)$ the infinity norm ball centered at c with radius r, i.e., $\mathcal{B}_r(c) = \{x \in \mathbb{R}^n \mid ||c - x||_\infty \leq r\}$.

Aircraft Dynamical Model. In this paper, we will consider an aircraft landing on a runway. We assume the states of the aircraft to be measured with respect to the origin of the Runway Coordinate Frame (shown in Fig. 1(left)), where positions are: ξ_x is the axis across runway; ξ_y is the altitude, and ξ_z is the axis along runway. We consider only one angle ξ_θ which represents the pitch rotation around x axis of the aircraft. The state vector of the aircraft at time $t \in \mathbb{N}$ is denoted by $\xi^{(t)} \in \mathbb{R}^4 = [\xi_\theta^{(t)}, \xi_x^{(t)}, \xi_y^{(t)}, \xi_z^{(t)}]^T$ and is assumed to evolve over time while being governed by a general nonlinear dynamical system of the form $\xi^{(t+1)} = f(\xi^{(t)}, u^{(t)})$ where $u^{(t)} \in \mathbb{R}^m$ is the control vector at time t. Such nonlinear dynamical system is assumed to be time-sampled from an underlying continuous-time system with a sample time equal to τ.

Runway Parameters. We consider runway that consists of two line segments L and R. Each line segment can be characterized by its start and end point (measured also in the Runway Coordinate Frame) i.e. $L = [(L_x, 0, L_z), (L_x, 0, L_z + r_l)]$ and $R = [(R_x, 0, R_z), (R_x, 0, R_z + r_l)]$, with $R_x = L_x + r_w$ and $R_z = L_z$ where r_w

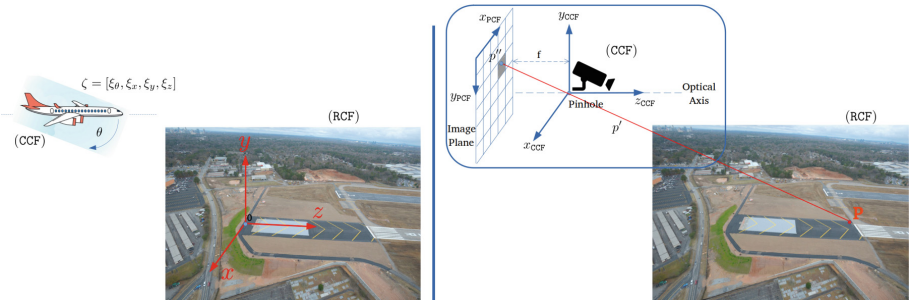

Fig. 1. Main coordinate frames: Runway (RCF), Camera (CCF) and Pixel (PCF).

and r_l refers to the runway width and length (standard international runways are designed with $r_w = 40$ m wide and $r_l = 3000$ m).

Camera Model. We assume the aircraft is equipped with a monochrome camera $\mathcal{C}$ that produces an image I of $q \times q$ pixels. Since the camera is assumed to be monochromatic, each pixel in the image I takes a value of 0 or 1. The image produced by the camera depends on the relative location of the aircraft with respect to the runway. In other words, we can model the camera $\mathcal{C}$ as a function that maps aircraft states into images, i.e., $\mathcal{C} : \mathbb{R}^4 \to \mathbb{B}^{q \times q}$. Although the images created by the camera depend on the runway parameters, for ease of notation, we drop this dependence from our notation in $\mathcal{C}$.

We utilize an ideal pinhole camera model [21] to capture the image formation process of this camera. In general, a point $p = (p_x, p_y, p_z)$ in the Runway Coordinate Frame (RCF) is mapped into a point $p' = (p'_{x_{\text{CCF}}}, p'_{y_{\text{CCF}}}, p'_{z_{\text{CCF}}})$ on the Camera Coordinate Frame (CCF) using a translation and rotation transformations defined by [13]:

$$\begin{bmatrix} p'_{x_{\text{CCF}}} \\ p'_{y_{\text{CCF}}} \\ p'_{z_{\text{CCF}}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & \cos\theta & \sin\theta & y \\ 0 & -\sin\theta & \cos\theta & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{1}$$

The camera then converts the 3-dimensional point p' on the camera coordinate frame into two-dimensional point p'' on the Pixel Coordinate Frame (PCF) as:

$$p'' = (p''_{x_{\text{PCF}}}, p''_{y_{\text{PCF}}}) = \left(\left\lfloor \frac{q_{x_{\text{PCF}}}}{q_{z_{\text{PCF}}}} \right\rfloor, \left\lfloor \frac{q_{y_{\text{PCF}}}}{q_{z_{\text{PCF}}}} \right\rfloor \right) \tag{2}$$

where:

$$\begin{bmatrix} q_{x_{\text{PCF}}} \\ q_{y_{\text{PCF}}} \\ q_{z_{\text{PCF}}} \end{bmatrix} = \begin{bmatrix} \rho_w & 0 & u_0 \\ 0 & -\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p'_{x_{\text{CCF}}} \\ p'_{y_{\text{CCF}}} \\ p'_{z_{\text{CCF}}} \\ 1 \end{bmatrix} \tag{3}$$

and f is the focal length of the camera lens, W is the image width (in meters), H is the image width (in meters), WP is the image width (in pixels), HP is the image height (in pixels), and $u_0 = 0.5 \times \text{WP}$, $v_0 = 0.5 \times \text{HP}$, $\rho_w = \frac{\text{WP}}{\text{W}}$, $\rho_h = \frac{\text{HP}}{\text{H}}$.

What is remaining is to map the coordinates of $p'' = \left(p''_{x_{\texttt{PCF}}}, p''_{y_{\texttt{PCF}}}\right)$ into a binary assignment for the different $q \times q$ pixels. But first, we need to check if p'' is actually inside the physical limits of the Pixel Coordinate Frame (PCF) by checking:

$$\text{visible} = \begin{cases} \text{yes} & |p''_{x_{\texttt{PCF}}}| \leq \frac{W}{2} \ \vee \ |p''_{y_{\texttt{PCF}}}| \leq \frac{H}{2} \\ \text{no} & \text{otherwise} \end{cases} \tag{4}$$

Whenever the point p'' is within the limits of PCF, then the pixel $I[i, j]$ should be assigned to 1 whenever the index of the pixel matches the coordinates $\left(p''_{x_{\texttt{PCF}}}, p''_{y_{\texttt{PCF}}}\right)$, i.e.:

$$I[i,j] = \begin{cases} 1 & (p''_{x_{\texttt{PCF}}} == i-1) \wedge (p''_{y_{\texttt{PCF}}} == j-1) \wedge \text{visible} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

for $i, j \in (1, 2, 3...\text{WP})$. Where for simplicity, we set HP = WP for square images. This process of mapping a point p in the Runway Coordinate Frame (RCF) to a pixel in the image I is summarized in Fig. 1 (right).

Neural Network Controller. The aircraft is controlled by a vision based neural network $\mathcal{NN}$ controller that maps the images I created by the camera C into a control action, i.e., $\mathcal{NN} : \mathbb{B}^{q \times q} \rightarrow \mathbb{R}^m$. We confine our attention to neural networks that consist of multiple layers and where Rectified Linear Unit (ReLU) are used as the non-linear activation units.

Problem Formulation. Consider the closed-loop vision based system Σ defined as:

$$\Sigma : \left\{ \ \xi^{(t+1)} = f(\xi^{(t)}, \mathcal{NN}(\mathcal{C}(\xi^{(t)}))).\right.$$

A trajectory of the closed loop system Σ that starts from the initial condition ξ_0 is the sequence $\{\xi^{(t)}\}_{t=0,\xi^{(0)}=\xi_0}^{\infty}$. Consider also a set of initial conditions $\mathcal{X}_0 \subset \mathbb{R}^4$. We denote by $\Sigma^{\mathcal{X}_0}$ the trajectories of the system Σ that starts from $\mathcal{X}_0$, i.e.,

$$\Sigma^{\mathcal{X}_0} = \bigcup_{\xi_0 \in \mathcal{X}_0} \{\xi^{(t)}\}_{t=0,\xi^{(0)}=\xi_0}^{\infty}.$$

We are interested in checking if the closed-loop system meets some specifications that are captured using Linear Temporal Logic (LTL) (or a Bounded-Time LTL) formula φ. Examples of such formulas may include, but are not limited to:

- $\varphi_1 := \Diamond\{\xi_\theta = 0 \wedge \xi_y = 0\}$ which means that the aircraft should *eventually* reach an altitude of zero while the pitch angle is also zero. Satisfying φ_1 ensures that the aircraft landed on the ground.
- $\varphi_2 := \Box\{\xi_z \leq 3000\}$ which ensures the aircraft will *always* land before the end of the runway (assuming a runway length that is equal to 3000 m).

For the formal definition of the syntax and semantics of LTL and Bounded-Time LTL formulas, we refer the reader to [5]. Given a formula φ that specifies correct landing, our objective is to design a bounded model checking framework that verifies if all the trajectories $\Sigma^{\mathcal{X}_0}$ satisfy φ (denoted by $\Sigma^{\mathcal{X}_0} \models \varphi$).

3 Framework

The verification problem described in Sect. 2 is challenging because it needs to take into account the nonlinear dynamics of the aircraft f, the image formation process captured by the camera model $\mathcal{C}$, and the neural network controller $\mathcal{NN}$.

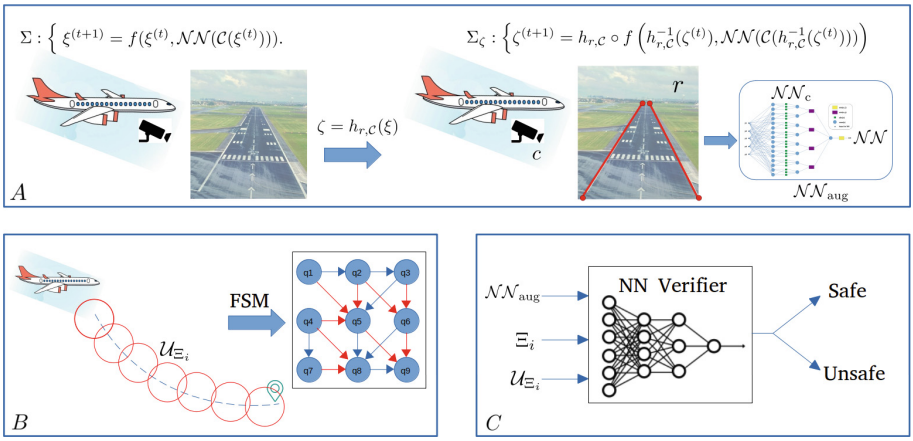

Fig. 2. Main elements of the proposed NNLander-VeriF framework: (A): construction of the augmented neural network that captures both perception and control, (B:) symbolic analysis of aircraft trajectories, (C:) neural network verification.

Our framework starts by re-modeling the pinhole camera model as a ReLU based neural network (with manually designed weights) that we refer to as the perception neural network $\mathcal{NN}_\mathcal{C}$. To facilitate this re-modeling, we need first to apply a change of coordinates to the states of the dynamical systems. We refer to the states in the new coordinates as ζ, i.e., $\zeta = h(\xi)$. By augmenting $\mathcal{NN}_\mathcal{C}$ along with the neural network controller $\mathcal{NN}$, one can obtain an augmented neural network $\mathcal{NN}_{\text{aug}} : \mathbb{R}^n \to \mathbb{R}^m$ defined as $\mathcal{NN}_{\text{aug}} = \mathcal{NN} \circ \mathcal{NN}_C$ and a simplified closed-loop dynamics, in the new coordinates, written as:

$$\Sigma : \{ \zeta^{(t+1)} = g(\zeta^{(t)}, \mathcal{NN}_{\text{aug}}(\zeta^{(t)})).$$

Now, assume that we are given (i) a region Ξ in the new coordinate system and (ii) the maximal set of control actions (denoted by $\mathcal{U}_\Xi$) that can be applied at Ξ while ensuring the system adhere to the specification φ. Given this pair $(\Xi, \mathcal{U}_\Xi)$ one can always ensure that the augmented neural network $\mathcal{NN}_{\text{aug}}$ will produce actions in the set $\mathcal{U}_\Xi$ whenever its inputs are restricted to Ξ by checking the following property:

$$\forall \zeta \in \Xi . \left(\mathcal{NN}_{\text{aug}}(\zeta) \in \mathcal{U}_\Xi \right) \tag{6}$$

which can be easily verified using existing neural network model checkers [10,18, 20]. In other words, checking the augmented neural network against the property above ensures that all the images produced within the region Ξ will force the neural network controller $\mathcal{NN}$ to produce control actions that are within the set of allowable actions $\mathcal{U}_{\Xi}$.

To complete our framework, we need to partition the state-space into regions $(\Xi_1, \Xi_2, \ldots)$. Each region is a ball parametrized by a center ζ_i and a radius ϵ. For each region, our framework will compute the set of allowable control actions at each of these regions $(\mathcal{U}_{\Xi_1}, \mathcal{U}_{\Xi_2}, \ldots)$. Our framework will also parametrize each set $\mathcal{U}_{\Xi_i}$ as a ball with center c_i and radius μ_i, i.e., $\mathcal{U}_{\Xi_i} = \{u \in \mathbb{R}^4 | \ \|u - c_i\| \leq \mu_i\}$. The computations of the pairs $(\Xi_i, \mathcal{U}_{\Xi_i})$ can be carried out using the knowledge of the aircraft dynamics f. In summary, and as shown in Fig. 2, our framework will consist of the following steps:

- **(A) Compute the augmented neural network**: Using the physical model of the pinhole camera, our framework will re-model the pinhole camera $\mathcal{C}$ as a neural network that can be augmented with the neural network controller to produce a simpler model that is amenable for verification.
- **(B) Compute the set of allowable control actions**: We use the properties of the dynamical system f to compute the set of safe control actions $\mathcal{U}_{\Xi_i}$ for each partition Ξ_i of the state space.
- **(C) Apply the neural network model checker:** We use the neural network model checkers to verify that $\mathcal{NN}_{\text{aug}}$ satisfies (6) for each identified pair $(\Xi, \mathcal{U}_{\Xi_i})$.

The remainder of this paper is devoted to providing details for the steps required for each of the three phases above.

4 Neural Network Augmentation

In this section, we focus on the problem of using the geometry of the runway to develop a different mathematical model for the camera $\mathcal{C}$. As argued in the previous section and shown in Fig. 3, our goal is to obtain a model with the same structure of a neural network (i.e., consists of several layers and neurons) and contains only ReLU activation units. We refer to this new model as $\mathcal{NN}_{\mathcal{C}}$.

The main challenge to construct $\mathcal{NN}_{\mathcal{C}}$ is the fact that ReLU based neural networks can only represent piece-wise affine (or linear) functions [22]. Nevertheless, the camera model $\mathcal{C}$ is inherently nonlinear due to the optical projection present in any camera. Such non-linearity can not be expressed (without any error) via a piece-wise affine function. To solve this problem, we propose a change of coordinates to the aircraft states h. Such change of coordinates is designed to eliminate part of the camera's non-linearity while allowing the remainder of the model to be expressed as a piece-wise affine transformation.

Change of Coordinates: Recall the runway consists of line segments L and R (defined in Sect. 2). Instead of measuring the state of the aircraft by the

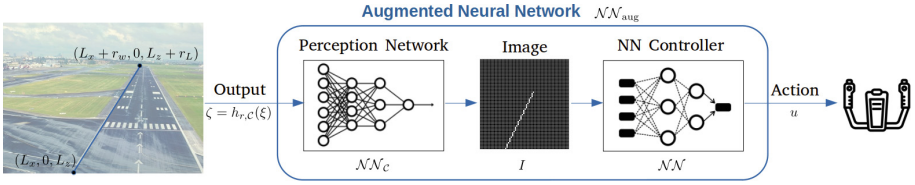

Fig. 3. Augmented network $\mathcal{NN}_{\text{aug}}$ maps the output ζ to control action u.

vector $\zeta = [\xi_\theta, \xi_x, \xi_y, \xi_z]$, we propose measuring the state of the aircraft by the projections of the end points of the lines L and R on the Pixel Coordinate Frame PCF. Formally, we define the change of coordinates as:

$$\zeta = h_{r,\mathcal{C}}(\xi) = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \\ \zeta_5 \end{bmatrix} = \begin{bmatrix} \rho_w f \frac{L_x + \xi_x}{L_z cos(\xi_\theta) + \xi_z} + u_0 \\ -\rho_h f \frac{L_z sin(\theta) + \xi_y}{L_z cos(\xi_\theta) + \xi_z} + v_0 \\ \rho_w f \frac{L_x + \xi_x}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + u_0 \\ -\rho_h f \frac{(L_z + r_L) sin(\xi_\theta) + \xi_y}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + v_0 \\ \zeta_1 \zeta_4 - \zeta_2 \zeta_3 \end{bmatrix} \tag{7}$$

where $f, \rho_h, \rho_w, v_0, u_0$ are the camera physical parameters as defined in Sect. 2. In other words, the pair (ζ_1, ζ_2) is the projection of the start point of the runway $(L_x, 0, L_z)$ onto the Pixel Coordinate Frame PCF (while ignoring the flooring operator for now). Similarly, the pair (ζ_3, ζ_4) is the projection of the endpoint of the runway $(L_x, 0, L_z + r_L)$ onto the PCF frame. Indeed, we can define a similar set of variables for the other line segment of the runway, R. The dependence of this change of coordinates on the camera parameters (e.g., the focal length f) and the runway parameters justifies the subscripts in our notation $h_{r,\mathcal{C}}$. We refer to the new state-space as Ξ.

Before we proceed, it is crucial to establish the following result.

Proposition 1. *The change of coordinates function $h_{r,\mathcal{C}}$ is bijective.*

The proof of such proposition is based on ensuring that the inverse function $h^{-1}_{r,\mathcal{C}}$ exists. For brevity, we will omit the details of this proof. Since $h_{r,\mathcal{C}}$ is bijective, we can re-write the closed-loop dynamics of the system as:

$$\Sigma_\zeta : \left\{ \zeta^{(t+1)} = h_{r,\mathcal{C}} \circ f \left(h^{-1}_{r,\mathcal{C}}(\zeta^{(t)}), \mathcal{NN}(\mathcal{C}(h^{-1}_{r,\mathcal{C}}(\zeta^{(t)}))) \right) \right. \tag{8}$$

Indeed, if Σ_ζ satisfies the property φ then do the original system Σ and vice versa, thanks for the fact that $h_{r,\mathcal{C}}$ is bijective. This is captured by the following proposition:

Proposition 2. *Consider the dynamical systems Σ and Σ_ζ. Consider a set of initial states $\mathcal{X}_0$ and an LTL formula φ, the following holds:*

$$\Sigma^{\mathcal{X}_0} \models \varphi \iff \Sigma_\zeta^{\Xi_0} \models \varphi$$

where $\Xi_0 = \{h_{r,\mathcal{C}}(\xi) \mid \xi \in \mathcal{X}_0\}$.

Neural Network-Based Model for Perception: While the model of the pinhole camera (defined in Eq. (1)–(5)) focuses on mapping individual points into pixels, we aim here to obtain a model that maps the entire runway lines R and L into the corresponding binary assignment for each pixel in the image. Therefore, it is insufficient to analyze the values of $\zeta_1, \ldots, \zeta_4$ which encodes the start point (ζ_1, ζ_2) and the endpoint (ζ_3, ζ_4) of the runway line segments on the PCF. To correctly generate the final image $I \in \mathbb{B}^{q \times q}$, we need to map *every* point between (ζ_1, ζ_2) and (ζ_3, ζ_4) into the corresponding pixels.

While the pinhole camera (defined in Eq. (1)–(4)) uses the information in the Pixel Coordinate Frame (PCF) to compute the values of each pixel, we instead rely on the information in the Camera Coordinate Frame (CCF) to avoid the nonlinearities added by the flooring operator in (2) and the logical checks in (4)–(5). For each pixel, imagine a set of four line segments AB, BC, CD, DA in the Pixel Coordinate Frame (PCF) that defines the edges of each pixel (see Fig. 4 for an illustration). To check if a pixel should be set to zero or one, it is enough to check the intersection between the line segment $(\zeta_1, \zeta_2) - (\zeta_3, \zeta_4)$ and each of the lines A–B, B–C, C–D, D–A. Whenever an intersection occurs, the pixel should be assigned to one.

To intersect one of the pixel edges, e.g., the edge $A - B = (A_x, A_y) - (B_x, B_y)$, with the line segment $(\zeta_1, \zeta_2) - (\zeta_3, \zeta_4)$, we proceed with the standard line segment intersection algorithm [6] which compute four values named O_1, O_2, O_3, O_4 as:

$$O_1 = \zeta_1(A_y - B_y) + \zeta_2(B_x - A_x) + A_x B_y - A_y B_x \tag{9}$$
$$O_2 = \zeta_3(A_y - B_y) + \zeta_4(B_x - A_x) + A_x B_y - A_y B_x \tag{10}$$
$$O_3 = -\zeta_1(A_y) + \zeta_2(A_x) + \zeta_3(A_y) - \zeta_4(A_x) + \zeta_5 \tag{11}$$
$$O_4 = -\zeta_1(B_y) + \zeta_2(B_x) + \zeta_3(B_y) - \zeta_4(B_x) + \zeta_5 \tag{12}$$

The line segment algorithm [6] detects an intersection whenever the following condition holds:

$$(\text{sign}(O_1) \neq \text{sign}(O_2)) \ \wedge \ (\text{sign}(O_3) \neq sign(O_4)) \tag{13}$$

Luckily, we can organize the equations (9)–(13) in the form of a neural network with a Rectifier Linear Activation Unit (ReLU). ReLU nonlinearity takes the form of $\text{ReLU}(x) = \max\{x, 0\}$. To show this conversion, we first note that the values of A_x, A_y, B_x, B_y are constant and well defined for each pixel. So assuming the input to such a neural network is the vector ζ, one can use equations (9)–(12) to assign the weights to the input layer of the neural network (as shown

in Fig. 4). To check the signs of $O_1, \ldots, O_4$, we recall the well-known identity for numbers of the same sign:

$$\text{sign}(a) = \text{sign}(b) \iff |a+b| - |a| - |b| = 0 \tag{14}$$

The absolute function can be implemented directly with a ReLU using the identity:

$$|x| = \max\{x, 0\} + \max\{-x, 0\}. \tag{15}$$

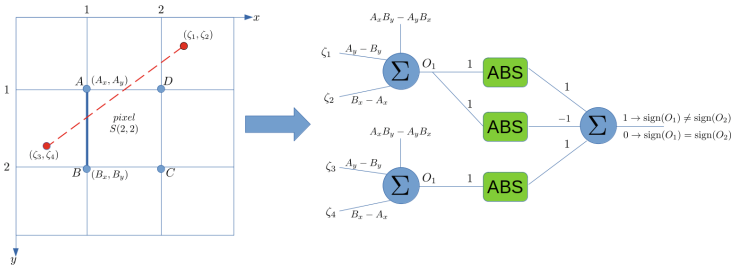

Fig. 4. Line-segment intersection algorithm: The runway line (in red) as seen by the camera intersects the pixel edge A–B (in blue), this single edge intersection is detected by using a layer of six ReLU's. (Color figure online)

The process above has to be repeated four times (to account for all edges A–B, B–C, C–D, D–A of a pixel). Finally, to check that at least one intersection occurred, we compute the minimum across the results from all the intersections. Calculating the minimum itself can be implemented directly with a ReLU using the identity:

$$min\{a, b\} = \frac{a+b}{2} - \frac{|a-b|}{2}. \tag{16}$$

The overall neural network requires $68 \times q \times q$ ReLU neurons for each projected line segment. The final architecture is shown in Fig. 5. We refer to the resulting neural network as $\mathcal{NN}_{\mathcal{C}}(\zeta^{(t)})$.

It is direct to show that the constructed neural network $\mathcal{NN}_{\mathcal{C}}(\zeta^{(t)})$ will produce the same images obtained by the pinhole camera model $\mathcal{C}$, i.e.,

$$\mathcal{C}(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)})) = \mathcal{NN}_{\mathcal{C}}(\zeta^{(t)})$$

Finally, by substituting in (8), we can now re-write the closed-loop dynamics as:

$$\Sigma_\zeta : \left\{ \zeta^{(t+1)} \quad = g\left(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)}), \mathcal{NN}_{\text{aug}}(\zeta^{(t)})\right) \right. \tag{17}$$

where $\mathcal{NN}_{\text{aug}} = \mathcal{NN} \circ \mathcal{NN}_{\mathcal{C}}$ and $g = h_{r,\mathcal{C}} \circ f$.

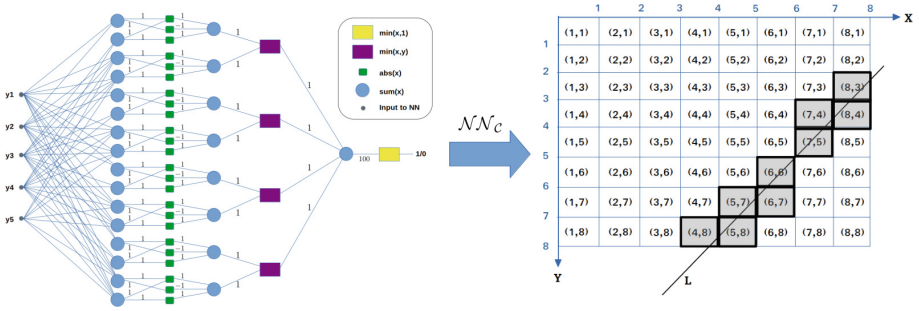

Fig. 5. $\mathcal{NN}_\mathcal{C}$ checks the intersection between line segment $(\zeta_1, \zeta_2) - (\zeta_3, \zeta_4)$ and all edges of each cell pixel of the final image.

5 Identifying the Allowable Control Actions Using Symbolic Abstractions

As shown in Sect. 3, our framework aims to split the verification of the dynamical system (17) into several NN model checking queries. Each query will verify the correctness of the closed-loop system within a region (or a symbol) Ξ_i of the state space. To prepare for such queries, we need to compute a set of input/output pairs $(\Xi_i, \mathcal{U}_{\Xi_i})$ with the guarantee that all the control inputs inside each $\mathcal{U}_{\Xi_i}$ will produce trajectories that satisfy the specifications φ. In this section, we provide details of how to compute the pairs $(\Xi_i, \mathcal{U}_{\Xi_i})$.

State Space Partitioning: Given a partitioning parameter ϵ, we partition the new coordinate space of ζ into L regions $\Xi_1, \Xi_2, \ldots, \Xi_L$ such that each Ξ_i is an infinity-norm ball with radius ϵ and center c_i. For simplicity of notation, we keep the radius ϵ constant within all the regions Ξ_i. However, the framework is generic enough to account for multi-scale partitioning schemes similar to those reported in the literature of symbolic analysis of hybrid systems [15].

Obtain Symbolic Models: Given the regions $\Xi_1, \Xi_2, \ldots$, the next step is to construct a *finite-state abstraction* for the closed loop system (17). Such finite state abstraction takes the form of a finite state machine $\Sigma_q = (S_q, \sigma_q)$ where S_q is the set of finite states and $\sigma_q : S_q \to 2^{S_q}$ is the state transition map of the finite state machine, defined as:

$$S_q = \{1, 2, \ldots L\} \quad \text{and} \quad j \in \sigma_q(i) \iff g\left(h_{r,\mathcal{C}}^{-1}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right) \in \Xi_j. \tag{18}$$

In other words, the finite state machine (FSM) has a number of states L that is equal to the number of regions Ξ_i, i.e., each finite state symbolically represents a region. A transition between the state i and j is added to the state transition map σ_q whenever applying the NN controller to the center of the region i (i.e., c_i) will force the next state of the system to be within the region Ξ_j. The value of $g\left(h_{r,\mathcal{C}}^{-1}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right)$ can be directly computed by evaluating the neural network $\mathcal{NN}_{\text{aug}}$ at the center c_i followed by evaluating the function g.

So far, the state transition map σ_q accounts only for actions taken at the center of the region. To account for the control actions in all the states $\zeta_i \in \Xi_i$, we need to bound the distance between the trajectories that start at the center of the region c_i and the trajectories that start from any other state $\zeta_i \in \Xi_i$. For such bound to exist, we enforce an additional assumption on the dynamics of the aircraft model f (and hence $g = h_{r,\mathcal{C}} \circ f$) named δ forward complete (δ-FC) [28]. Given the center of a region c_i and an arbitrary state $\zeta_i \in \Xi_i$, the δ-FC assumption bounds the distance, denoted by δ_ζ, between the trajectories that starts at ζ_i and the center c_i as:

$$\delta_\zeta \leq \beta(\epsilon, \tau) + \gamma(||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty, \tau) \tag{19}$$

where τ is the sample time used to obtain the dynamics f (as explained in Sect. 2) and β and γ are class K_∞ functions that can be computed from the knowledge of the dynamics f. Such δ-FC assumption is shown to be mild and does not require the aircraft dynamics to be stable. For technical details about the δ-FC assumption and the computation of the functions β and γ, we refer the reader to [27]. Given the inequality (19), we can revisit the definition of the state transition map σ_q to account for all possible trajectories as:

$$j \in \sigma_q(i) \iff g\left(h^{-1}_{r,\mathcal{C}}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right) + \delta_\zeta \in \Xi_j. \tag{20}$$

With such a modification, it is direct to show the following result:

Proposition 3. *Consider the dynamical systems Σ_ζ and Σ_q. Consider also a set of initial conditions Ξ_0 and a specification φ. The following holds:*

$$\Sigma_q^{\mathcal{S}_0} \models \varphi \Rightarrow \Sigma_\zeta^{\Xi_o} \models \varphi$$

where $\mathcal{S}_0 = \{i \in \{1, \ldots, L\} \mid \exists\, \zeta_0 \in \Xi_0 : \ \zeta_0 \in \Xi_i\}$.

This proposition follows directly from Theorem 4.1 in [28].

Compute the Set of Allowable Control Actions: Unfortunately, computing the norm $||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty$ (and hence δ_ζ) is challenging. As shown in [16], computing such norm is NP-hard and existing tools in the literature focus on computing an upper bound for such norm. Nevertheless, the bounds given by the existing literature constitute large error margins that will render our approach severely conservative.

To alleviate the problem above, we use the inequality (19) in a "backward design approach". We first search for the maximum value of δ_ζ that renders Σ_q compatible with the specification. To that end, we substitute the norm $||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty$ with a dummy variable μ. By iteratively increasing the value of μ, we will obtain different Σ_q, one for each value of μ. We use a bounded model checker for each value of μ to verify if the resulting Σ_q satisfies the specification. We keep increasing the value of μ until the resulting Σ_q no longer satisfies φ. We refer to this value as $\mu_{\max}$. What is remaining is to ensure that the neural network indeed respects the bound:

$$||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty \leq \mu_{\max}$$

Algorithm 1. LanderNN-VeriF

Input: $\Xi, \Xi_0, \varphi, \epsilon, \tau, \beta, \gamma, \mathcal{NN}_{\text{aug}}, T, \overline{\mu}, \underline{\mu}, f, h, h^{-1}$
Output: `STATUS`

$\{\Xi_1, \Xi_2, \ldots, \Xi_L\} =$ `Partition_into_regions`(Ξ, ϵ)
$\mu = \underline{\mu}$
while `statusFSM` == UNSAT **do**
  $\Sigma_q =$ `Create_FSM`$(f, h, h^{-1}, \tau, \beta, \gamma, \mathcal{NN}_{aug}, \Xi_{1..L}, \mu)$
  `statusFSM` = `Check_FSM`(φ, Σ_q, T)
  if $\mu \leq \overline{\mu}$ **then**
    $\mu =$ `Increase_MU`(μ)
  end if
end while
for i = 1 to L **do**
  `STATUS_NN[i]` = `NN_Verifier`$(\mathcal{NN}_{\text{aug}}, \Xi_i, \mu)$
  if `STATUS_NN[i]` == SAT **then**
    `STATUS` = UNSAFE
  else
    `STATUS` = SAFE
  end if
end for
return `STATUS`

To that end, we define the set of allowable control actions $\mathcal{U}_{\Xi_i}$ as:

$$\mathcal{U}_{\Xi_i} = \mathcal{B}_{\mu_{\max}}(\mathcal{NN}_{\text{aug}}(c_i))$$

It is then direct to show the following equivalence:

$$||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty \leq \mu_{\max} \iff \forall \zeta \in \Xi_i.\big(\mathcal{NN}_{\text{aug}}(\zeta) \in \mathcal{U}_{\Xi_i}\big)$$

where $\mathcal{U}_{\Xi_i} = \mathcal{B}_{\mu_{\max}}(\mathcal{NN}_{\text{aug}}(c_i))$. Luckily, the right-hand side of this equivalence is precisely what neural network model checkers are capable of verifying. Algorithm 1 summarizes this discussion. The following result captures the guarantees provided by the proposed framework:

Proposition 4. *The LanderNN-VeriF algorithm (Algorithm 1) is sound but not complete.*

6 Numerical Example

We illustrate the results in this paper using a vision-based aircraft landing system. We use a fixed-wing aircraft model defined using the guidance kinematic model [3], where orientations (in Rads) are defined by the course angle χ (rotation around y_{CCF} axis), pitch angle θ (rotation around x_{CCF} axis) and V_g denotes the total Aircraft velocity relative to the ground. We further simplify the system by keeping the course angle pointing towards the runway ($\chi = 0$), similarly velocity is kept as constant. Moreover, $\dot{\theta}$ (Rad/s) is regarded as the control

input u. According to this model, the state vector of the aircraft evolves over time while being governed by the following dynamical system [3]:

$$\xi_z^{(t+1)} = \xi_z^{(t)} + V_g \tau \cos(\xi_\theta^{(t)}) \tag{21}$$

$$\xi_y^{(t+1)} = \xi_y^{(t)} + V_g \tau \sin(\xi_\theta^{(t)}) \tag{22}$$

$$\xi_\theta^{(t+1)} = \xi_\theta^{(t)} + u^{(t)} \tau \tag{23}$$

where τ is the sampling time. For our simulations we consider $V_g = 25\frac{m}{s}$ and $\tau = 0.1$. Moreover based on airport standards we consider the runway segments (in meters) defined by $L = [(L_x, 0, L_z), (L_x, 0, L_z + r_l)]$ and $R = [(R_x, 0, R_z), (R_x, 0, R_z + r_l)]$ where $R_x = 20$, $L_x = -20$, $R_z = 0$, $L_z = 0$, $r_l = 3000$. For the camera parameters we consider images of 16×16 pixels and focal length of 400 mm.

We note that the system dynamics (21)–(23) is a δ-FC system. In particular, by using the method [27] and the δ-FC Lyapunov function $\mathcal{V}(\xi, \xi') = ||\xi - \xi'||_2^2$ one can show that:

$$\beta(\zeta_1, \zeta_2, \zeta_3, \tau) = \sqrt{8}\sqrt{\zeta_1^2 + \zeta_2^2 + \zeta_3^2}\, e^{\tau} \tag{24}$$

$$\gamma(\mu, \tau) = \sqrt{V_g(e^{2\tau} - 1)}\mu \tag{25}$$

We work on the output space set $D = [\zeta_1 \times \zeta_2 \times \zeta_3] = [0, 16] \times [0, 16] \times [0, 16]$ of Σ_ζ with a precision $\epsilon = 1$, thus our discretized grid consists of 16^3 cubes.

We used Imitation Learning to train a fully connected ReLU Neural Network controller ($\mathcal{NN}$) of 2 layers with 128 Neurons each. Trajectories from different initial conditions were collected and used to train the network. Our objective is to verify that the aircraft landing using the trained $\mathcal{NN}_{\text{aug}}$ satisfies the safety specification $\phi = \Box\neg q_{\text{unsafe}}$ where $q_{\text{unsafe}} = [\xi_z = 800, \xi_y = 200, \xi_\theta = 1]$ which corresponds to an unsafe region while landing.

In what next, we report the execution time to verify the trained network. All experiments were executed on an Intel Core i7 processor with 50 GB of RAM. First, we implemented our Vision Network ($\mathcal{NN}_{\mathcal{C}}$) for images of 16×16 pixels using Keras. Similarly, we used Keras composition libraries to merge the controller and perception networks into the augmented network ($\mathcal{NN}_{\text{aug}}$), a landing trajectory using $\mathcal{NN}_{aug}$ is shown in Fig. 6 and its corresponding camera view is shown in Fig. 7.

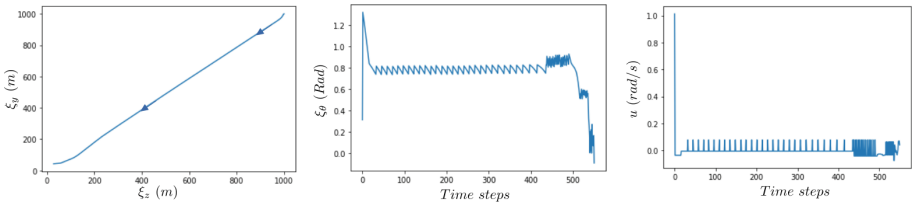

Fig. 6. Aircraft landing using augmented controller $\mathcal{NN}_{aug}$. Left: aircraft position (ξ_y, ξ_z); Middle: aircraft angle (ξ_θ); Right: aircraft control $(u = \mathcal{NN}_{aug})$.

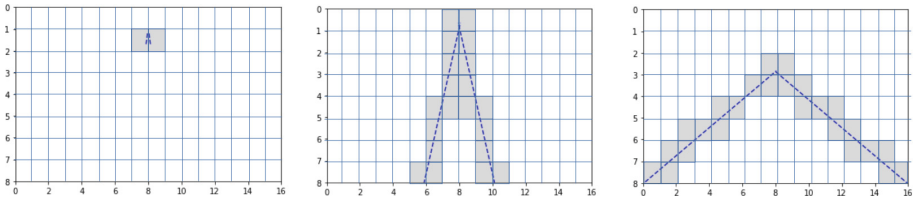

Fig. 7. Landing camera view using 16×16 pixels resolution. Left: $\xi^1 = [1000, 1000, \frac{\pi}{4}]$, Middle: $\xi^{300} = [400, 300, \frac{\pi}{8}]$, Right: $\xi^{1000} = [5, 5, 0]$.

We used a Boolean SAT solver named SAT4J [4] to implement the `Check_FSM` function in Algorithm 1. The finite state machine Σ_q was encoded using a set of Boolean variables and our implementation performed a bounded model checking for the generated FSMs (the bounded model checking horizon was set to 20). We constructed FSMs with the following values $\mu = [0.1, 0.2, 0.3, 0.6, 0.8, 0.9, 1.1]$ until a value of $\mu_{max} = 1.1$ was found. The execution time for creating Σ_q and verifying its properties with the bounded model checker increased monotonically from 2000 seconds for $\mu = 0.1$ to 7000 seconds for $\mu = 1.1$. As expected, the higher the value of μ, the higher the number of transitions in Σ_q, and the higher the time needed to create and verify.

Finally, we used PeregriNN [20] as the NN model checker. Figure 8 reports the execution time for verifying the neural network property in 100 random regions, and Fig. 9 in regions 1 to 500. The average execution time was 76 s per region and the NN was found to be safe and satisfying the specification φ.

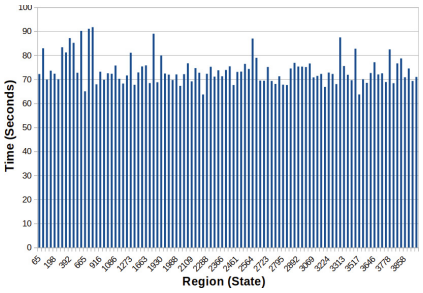

Fig. 8. Execution time for verifying φ in 100 different random regions.

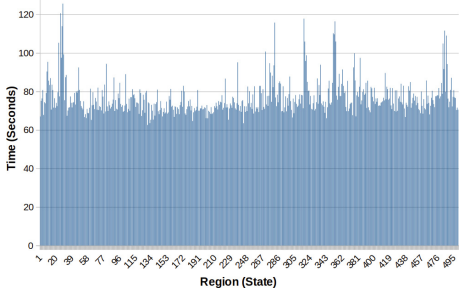

Fig. 9. Execution time for verifying φ in regions 1 to 500.

7 Conclusion and Future Work

Due to the recent surge in vision-based autonomous systems, it is becoming increasingly important to provide frameworks to facilitate its formal verification. In this work we have proposed two key contributions: first, a generative

model that encodes part of the camera image formation process into a ReLU neural network, where the neuronal weights are fully determined by the camera intrinsic parameters, and second, a framework that uses the characteristics of the dynamical system (i.e. δ-FC) to compute the set of safe control actions; Finally, having both contributions allows us to use off-the-shelf neural network checkers to verify the entire system.

At the same time, there are some limitations. First, the generative model we developed insists on modeling the image formation process with a piecewise affine (PWA) function which facilitates encoding it as a ReLU network. However, this restriction may in odds with realistic scenarios which may not be captured exactly by CPWA functions. Nevertheless, it is widely known that CPWA functions can approximate general nonlinear functions with some error. This also leads to the second limitation, namely, the inability to consider noise in the image formation process. Finally, the number of pixels has a direct effect on the scalability of the framework, as a consequence further improvements are required to build more concise finite-state machine abstractions of the physical system.

Moving forward, we plan to extend our approach in different directions to account for the aforementioned limitations. First, we seek to generalize the framework to account for uncertainties in the camera model, the image formation model, and the environment. Second, we intend to process more complex image features (e.g. combinations of multiple lines and curvatures) by developing better generative models with provable error bounds. Finally, we aim to verify the robustness of neural network controllers to external disturbances (e.g., wind) while developing better scalable algorithms.

References

1. International Verification of Neural Networks Competition 2020 (VNN-COMP 2020). https://sites.google.com/view/vnn20
2. Bak, S., Tran, H.-D., Hobbs, K., Johnson, T.T.: Improved Geometric path enumeration for verifying ReLU neural networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020, Part I. LNCS, vol. 12224, pp. 66–96. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_4
3. Beard, R.W., Mclain, T.W.: Small Unmanned Aircraft: Theory and Practice. Princeton University Press, Princeton (2012)
4. Berre, D.L., Parrain, A.: The Sat4j library. Boolean Model. Comput. **7**, 59–64 (2010)
5. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of Model Checking, vol. 10. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-10575-8
6. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2003)
7. Cruz, U.S., Ferlez, J., Shoukry, Y.: Safe-by-repair: a convex optimization approach for repairing unsafe two-level lattice neural network controllers. arXiv preprint arXiv:2104.02788 (2021)

8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
9. Faugeras, O., Faugeras, O.A.: Three-Dimensional Computer Vision: A Geometric Viewpoint. MIT Press, Cambridge (1993)
10. Ferlez, J., Khedr, H., Shoukry, Y.: Fast BATLLNN: fast box analysis of two-level lattice neural networks. In: Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control (2022)
11. Ferlez, J., Shoukry, Y.: Bounding the complexity of formally verifying neural networks: a geometric approach. In: 2021 60th IEEE Conference on Decision and Control (CDC), pp. 5104–5109. IEEE (2021)
12. Fremont, D.J., Chiu, J., Margineantu, D.D., Osipychev, D., Seshia, S.A.: Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020, Part I. LNCS, vol. 12224, pp. 122–134. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_6
13. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge (2003)
14. Hsieh, C., Joshi, K., Misailovic, S., Mitra, S.: Verifying controllers with convolutional neural network-based perception: a case for intelligible, safe, and precise abstractions. arXiv preprint arXiv:2111.05534 (2021)
15. Hsu, K., Majumdar, R., Mallik, K., Schmuck, A.K.: Multi-layered abstraction-based controller synthesis for continuous-time systems. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), pp. 120–129 (2018)
16. Kallus, N., Zhou, A.: Assessing disparate impact of personalized interventions: identifiability and bounds. Adv. Neural Inf. Process. Syst. **32** (2019)
17. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017, Part I. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
18. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019, Part I. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
19. Katz, S.M., Corso, A.L., Strong, C.A., Kochenderfer, M.J.: Verification of image-based neural network controllers using generative models. arXiv preprint arXiv:2105.07091 (2021)
20. Khedr, H., Ferlez, J., Shoukry, Y.: PEREGRiNN: penalized-relaxation greedy neural network verifier. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021, Part I. LNCS, vol. 12759, pp. 287–300. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_13
21. Ma, Y., Soatto, S., Kosecka, J., Sastry, S.S.: An Invitation to 3-D Vision: From Images to Geometric Models, vol. 26. Springer, Heidelberg (2012). https://doi.org/10.1007/978-0-387-21779-6
22. Nagamine, T., Mesgarani, N.: Understanding the representation and computation of multilayer perceptrons: a case study in speech recognition. In: International Conference on Machine Learning, pp. 2564–2573. PMLR (2017)
23. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 147–156 (2019)
24. Sun, X., Shoukry, Y.: Provably correct training of neural network controllers using reachability analysis. arXiv preprint arXiv:2102.10806 (2021)

25. Tran, H.-D., et al.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020, Part I. LNCS, vol. 12224, pp. 3–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
26. Wang, Y.S., Weng, L., Daniel, L.: Neural network control policy verification with persistent adversarial perturbation. In: International Conference on Machine Learning, pp. 10050–10059. PMLR (2020). https://proceedings.mlr.press/v119/wang20v.html
27. Zamani, M.: Control of cyber-physical systems using incremental properties of physical systems. Ph.D. thesis (2012)
28. Zamani, M., Pola, G., Mazo, M., Jr., Tabuada, P.: Symbolic models for nonlinear control systems without stability assumptions. IEEE Trans. Autom. Control **57**(7), 1804–1809 (2012)