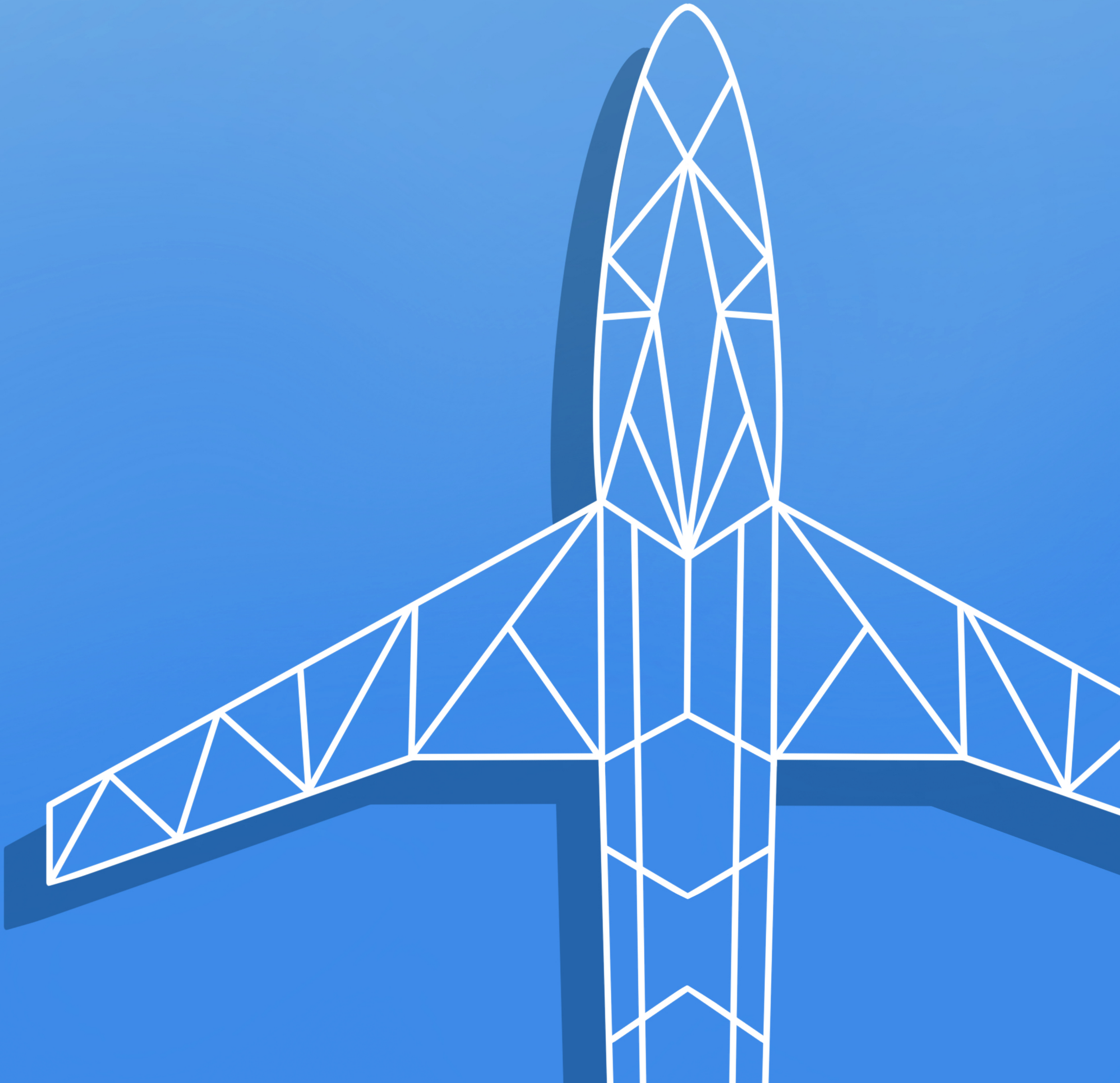
FORMAL VERIFICATION OF VISION-BASED AI-CONTROLLED CYBER-PHYSICAL VEHICLES

ULICES SANTA CRUZ LEAL



UNIVERSITY OF CALIFORNIA,
IRVINE

Formal Verification of Vision-Based AI-Controlled Cyber-Physical Vehicles

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Ulices Santa Cruz Leal

2024

DEDICATION

First and foremost, I dedicate this work to God, whose perfect wisdom has guided me every step of the way. In His infinite plan, not a single leaf falls without His permission. I am profoundly grateful for His constant presence and the faith that has sustained me, knowing that through Him, all things are possible.

I dedicate this thesis to my family. To my mother, Mireya; my father, David; my brother, David Jr.; and my grandparents, Ulises Fernandez and Cecia Rios. Your unwavering love and support have been my rock through every high and low. This work stands as a testament to the belief you instilled in me about the value of pursuing one's passion with discipline and faith in oneself.

But Jesus immediately said to them: "Take courage! It is I. Don't be afraid."
"Lord, if it's you" Peter replied, "tell me to come to you on the water."
"Come" he said.
Then Peter got down out of the boat, walked on the water, and came toward Jesus. But when he saw the wind, he was afraid and, beginning to sink, cried out, "Lord, save me!"
Immediately Jesus reached out his hand and caught him. "You of little faith" he said, "why did you doubt?"
And when they climbed into the boat, the wind died down. Then those who were in the boat worshiped him, saying, "Truly you are the Son of God."

Matthew 14:22-33

"But seek first the kingdom of God and his righteousness, and all these things will be added to you"

Matthew 6:33

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I'm profoundly grateful to my advisor, Prof. Yasser Shoukry, whose mentorship has been an irreplaceable treasure. Beyond imparting the essence of scientific inquiry, he nurtured discipline in me akin to a father's guidance. My enduring love for the realms of Control Theory, Artificial Intelligence, and Formal Methods is a testament to his extraordinary genius and expertise, which played a pivotal role in shaping my scientific aspirations. I owe an eternal debt of gratitude to him for his unwavering guidance and boundless inspiration.

I am also thankful to the committee members Prof. Yanning Shen and Prof. Solmaz Kia both geniouses and inspirations to me on the fields of Optimization and Controls for their meticulous review and insightful comments, which greatly enriched my work.

To my colleagues and friends at the Resilient Cyber-Physical Systems Lab: Wael Fatnassi, James Ferlez, Haitham Khedr, Momina Sajid, Kohei Tsujio, Goli Vaisi, Xiaowu Sun, Jae-jeong Park, and Mahmoud ElFar, your camaraderie provided a supportive environment amidst the challenges of a Ph.D. journey.

Special thanks to Ms. Luisa Kregel for her invaluable personal guidance throughout my Ph.D. journey. Her mentorship guided me through challenges that tested my will, illustrating that true growth is achieved through overcoming adversity. She encouraged me to honor my Mexican heritage while adapting to the United States of America and embracing its rich and independent cultural values.

I am deeply grateful to the scientific institutions CONACYT, UC MEXUS, and MGREP for their generous funding, which made this journey possible.

Last but not least, I want to express my heartfelt gratitude to my dear Taylor Pochron. The beautiful artwork she created for my thesis cover perfectly captures its essence. Her unwavering support and loving encouragement during the final stage of my journey have been incredibly meaningful, much like the cheers that carry a runner through the final stretch.

VITA

Ulices Santa Cruz Leal

EDUCATION

Ph.D. in Electrical & Computer Engineering	**2024**
University of California, Irvine	*Irvine, California, USA*
MS.c. in Astronautics & Space Engineering	**2018**
Cranfield University	*Bedford, England, UK*
BS.c. in Mechatronics Engineering	**2014**
Universidad Autónoma de Baja California	*Mexicali, México*

RESEARCH EXPERIENCE

Graduate Research Assistant	**2019–2024**
University of California, Irvine	*Irvine, California, USA*
Graduate Research Assistant	**2018**
Cranfield University	*Bedford, England, UK*

TEACHING EXPERIENCE

Teaching Assistant	**2023–2024**
University of California, Irvine	*Irvine, California, USA*

PROFESSIONAL EXPERIENCE

Solutions Engineer	**2024**
Applied Dynamics International	*Long Beach, California, USA*
Applied Scientist Intern	**2023**
Amazon AI Labs	*Pasadena, California, USA*
Quality Engineer	**2019**
Safran Electronics & Defense	*Mexicali, Mexico*
Test Engineer	**2015-2017**
Honeywell Aerospace	*Mexicali, Mexico*
Test Engineer Intern	**2014**
Max Planck Institute of Physics	*Munich, Germany*

REFEREED JOURNAL PUBLICATIONS

Certified Vision-based State Estimation using Geometric Generative Models **2024**
IEEE Transactions on Robotics (IEEE T-RO), Submitted.

Ulices Santa Cruz Leal, Mahmoud ElFar and Yasser Shoukry

REFEREED CONFERENCE PUBLICATIONS

Certified Vision-based State Estimation for Autonomous Landing Systems **Jul 2023**
62nd IEEE Conference on Decision and Control (CDC)

Ulices Santa Cruz Leal and Yasser Shoukry

NNLander-VeriF: A Neural Network Formal Verification Framework for Vision-Based Autonomous Aircraft Landing **Mar 2022**
14th NASA Formal Methods International Symposium

Ulices Santa Cruz Leal and Yasser Shoukry

Safe-by-Repair: A Convex Optimization Approach for Repairing Unsafe Two-Level Lattice Neural Network Controllers **Jul 2022**
61st IEEE Conference on Decision and Control (CDC)

Ulices Santa Cruz Leal, James Ferlez and Yasser Shoukry

Provably Safe Model-Based Meta Reinforcement Learning: An Abstraction-Based Approach **Jul 2021**
60th IEEE Conference on Decision and Control (CDC)

Xiaowu Sun, Wael Fatnassi, Ulices Santa Cruz Leal and Yasser Shoukry

Orbit modeling for simultaneous tracking and navigation using LEO satellite signals **Jul 2019**
32nd International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+ 2019)

Joshua J. Morales, Joe Khalife, Ulices Santa Cruz and Zaher M. Kassas

Networked and distributed cooperative attitude control of fractionated small satellites **Jul 2018**
69th International Astronautical Congress

Florian Kempf, Ulices Santa Cruz Leal, Julian Scharnagl and Klaus Schilling

ABSTRACT OF THE DISSERTATION

Formal Verification of Vision-Based AI-Controlled Cyber-Physical Vehicles

By

Ulices Santa Cruz Leal

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2024

Associate Professor Yasser Shoukry, Chair

The autonomous landing of small airplanes remains a significant challenge in aviation safety, with the Federal Aviation Administration (FAA) reporting an average of three crashes per day in the United States. While autonomous landing systems for large commercial airplanes exist, accessible technology for small airplanes is lacking, necessitating alternative solutions. Vision-based perception using cameras, coupled with artificial neural networks (NN), offers a promising approach by leveraging structural information to infer the vehicle's pose and surrounding environment, enabling safe and reliable landings. This research addresses key challenges in ensuring the safety and reliability of vision-based closed-loop systems for autonomous landings.

A central issue is the absence of robust mathematical models that accurately capture the relationship between system states, camera images, and the NN controller. Existing NN model checkers also fall short in verifying complex closed-loop systems. This Ph.D. thesis proposes NNLander-VeriF, a specialized framework for model-checking vision-based NN controllers in autonomous landing scenarios. The framework re-models the camera's physical model as a perception network, augmenting it with the NN controller to simplify closed-loop dynamics. This approach enables encoding closed-loop verification as robustness queries, leveraging state-of-the-art NN model checkers.

The research also focuses on designing certified vision-based state estimators using a generative NN model to enhance scene feature representation and aircraft state estimation. This model's mixed-monotonicity property aids in performing reachability analysis, providing guarantees on estimation accuracy and certifiable error bounds. Experimental results using real-world images from event-based cameras demonstrate significant improvements in state estimation accuracy and reliability.

Lastly, the research introduces a repair algorithm for NN controllers to address unsafe closed-loop behavior in certain states. By formulating the repair problem as convex optimization problems, the algorithm effectively repairs unsafe behavior while preserving safety for other states. The proposed framework enhances trust in autonomous landing systems through closed-loop verification, certified state estimation, and safe-by-repair techniques. This work aims to reduce crash rates in small airplanes, paving the way for safer autonomous landings and broader operational opportunities in aviation.

Chapter 1

Introduction

Autonomous landing of small airplanes presents a pressing challenge in the field of aviation safety, with an alarming frequency of crashes reported by the Federal Aviation Administration (FAA) averaging three per day in the United States alone. While autonomous landing systems have been developed for large commercial airplanes in international airports, the lack of accessible technology for small airplanes has necessitated exploring alternative solutions. To address this, vision-based perception, utilizing a camera to perceive the world, coupled with artificial neural networks (NN), has emerged as a promising approach. By leveraging the structural information captured by the camera, valuable insights can be inferred about the vehicle's pose and the surrounding scenario, enabling safe and reliable landings. However, several key challenges must be addressed to ensure the effectiveness of such vision-based systems in this context.

This research aims to address the safety and reliability of vision-based closed-loop systems in the context of aircraft autonomous landing using artificial intelligence. A central challenge lies in the absence of robust mathematical models that accurately capture the intricate relationship between system states, the images processed by a camera, and the vision-based

NN controller. Furthermore, existing NN model checkers are limited in their ability to reason about complex properties beyond simple input-output robustness. These limitations create a gap between the capabilities of the model checkers and the need to verify closed-loop systems while considering the dynamic behavior of the aircraft, the perception, and the NN controller. Current state-of-the-art artificial intelligence techniques, such as machine learning, relies on training a NN by using data (e.g., imitation learning) or reward functions (e.g., reinforcement learning). This is insufficient because it lacks mathematical guarantees in terms of correctness, generalization and interpretability. This becomes even more critical when the NN is used in a closed loop. This Ph.D. thesis proposes a design-verify-repair framework to design a correct-by-design NN as explained below.

Formal Verification of Perception Based AI Controllers: This research proposes NNLander-VeriF, an innovative and specialized framework explicitly tailored for model-checking vision-based NN controllers in autonomous landing scenarios. The framework leverages geometric models of perspective cameras to establish a comprehensive mathematical framework that accurately captures the intricate relationship between aircraft states and the inputs to the NN controller. The key contribution of the framework involves re-modeling the physical model of the camera as a neural network, which we refer to as the perception network. By augmenting the perception NN with the original NN controller, we create an augmented NN that simplifies the closed-loop dynamics. This augmentation enables us to encode closed-loop verification as a series of robustness queries, facilitating the use of state-of-the-art NN model checkers in the verification process.

To verify the performance of the augmented NN, the proposed framework partitions the state space into regions; each region is represented as a ball with a center and a radius. Within each region, the framework computes the set of allowable control actions by leveraging the properties of the dynamical system.

The framework then uses existing NN model checkers to ensure that the augmented NN satisfies the desired safety property (e.g., smooth aircraft landing). This property requires that for every state within a region, the augmented NN produces control actions within the corresponding set of allowable actions, providing an exhaustive formal verification of the vision-based autonomous landing system.

Training of Probably Correct Perception Based AI Models: In parallel to the verification aspect, this research also focused on the design of certified vision-based state estimators. Building upon the NNLander-VeriF framework, we introduced a certified NN-based generative model capable of producing N-polygon figures that describe the scene features and their relation to the aircraft states. Integrating this NN-based generative model within the training of the state estimator offers several advantages. Firstly, it provides a comprehensive and detailed representation of the scene features, allowing for a more accurate estimation of the aircraft's relative position with respect to the runway. By capturing the intricate details of the environment, the generative neural network enhances the system's understanding of the surrounding context, leading to more reliable and precise state estimation (e.g., 3D position and orientation of the aircraft with respect to the airport runway).

Moreover, this research showed that the certified generative model enjoys several mathematical properties, like the so-called mixed-monotonicity property. This benign property plays a crucial role in performing reachability analysis, allowing us to reason about the relationship between sets of aircraft states and sets of possible images. By leveraging this property, we can provide guarantees on the estimation accuracy and certifiable error bounds, enhancing the overall reliability and trustworthiness of the autonomous landing system. To validate the effectiveness of the proposed approach, comprehensive experiments are conducted using real-world images collected from event-based cameras. The experimental results demonstrate the significant improvement achieved in state estimation accuracy and the reliability of the generative neural network's output.

Repair of Faulty AI Models: Finally, this research focused on a critical challenge associated with repairing NN controllers. In real-world scenarios, NN controllers can exhibit unsafe closed-loop behavior in certain states, posing a significant risk of catastrophic consequences, such as an unavoidable aircraft crash given specific speed and position conditions. To tackle this issue, we introduced an innovative repair algorithm that effectively separates the local and global tension in modifying ReLU neural networks. This algorithm systematically and efficiently repairs the observed unsafe closed-loop behavior in known unsafe states, while ensuring closed-loop safety for a separate set of states. By formulating the repair problem as two distinct convex optimization problems, we successfully repaired a NN controller trained for an aircraft dynamical model, preserving the safety achieved during the initial data training phase.

In conclusion, the proposed framework enhances trust in autonomous landing systems by incorporating closed-loop verification, certified vision-based state estimation, and safe-by-repair techniques. This research tackles the challenge of high crash rates in small airplanes, paving the way for safer and more reliable autonomous landings. Further research in this direction could revolutionize aviation safety and expand opportunities for small airplane operations in different contexts, ensuring secure and efficient airspace.

Part I

Formal Verification of Perception Based AI Controllers

Chapter 2

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

In this chapter, 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 a perception-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-based components, and the NN controller.

To this end, this chapter 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 properties of a trained NN and we show its efficiency.

2.1 Introduction

Machine learning models, like deep neural networks, are used heavily to process high-dimensional imaging data like LiDAR scanners and cameras. These data 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 is the work on creating specialized decision procedures that can reason about input-output properties of NNs [26, 25, 9, 3, 31, 12, 14, 43, 45]. 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 system as a simple input-output property of NNs is still an open challenge.

On the other hand, *system-level* verification refer to the ability of reasoning about the temporal evolution of the whole system (including the NNs) while providing safety and liveness assurance. 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 [38]. While first steps were taken to provide formal models for LiDAR based systems [38], 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 [27, 21]. Unfortunately, these abstractions are only tested on a set of samples and lacks any formal guarantees in their ability to model the perception system formally. Other techniques uses 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 [17].

Motivated by the lack of formal guarantees of the abstractions of perception components [27, 21], we argue in this chapter 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 [32, 10]. 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 chapter, we employ our philosophy of data-driven design and model-based verification of vision-based autonomous systems to the problem of designing a NN that controls an aircraft while approaching runways to perform autonomous landing. Such problem enjoys geometric nature that can be exploited to develop 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 efficiently using the state-of-the-art component-level NN model checkers.
- We evaluated the proposed framework on a NN controller trained using imitation learning.

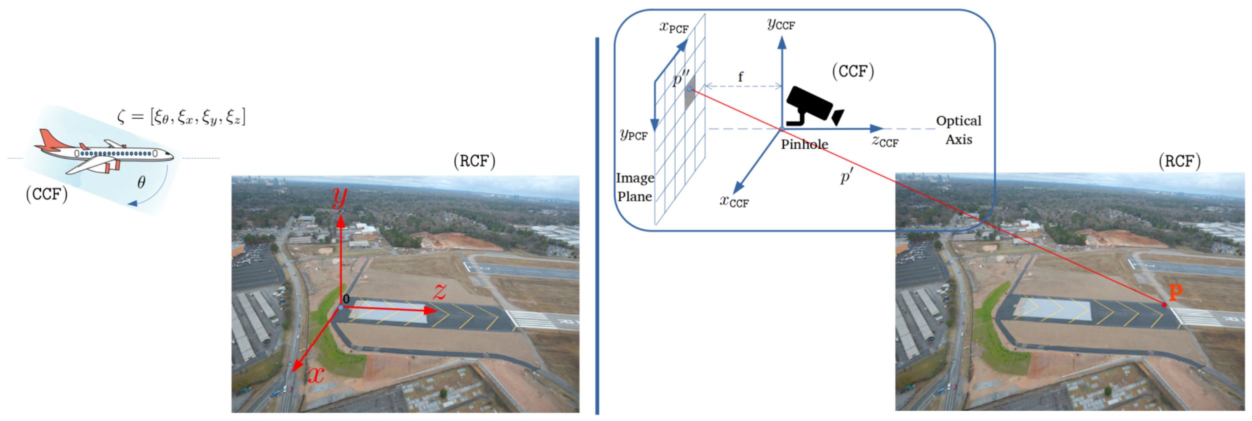


Figure 2.1: Main coordinate frames: Runaway (RCF), Camera (CCF) and Pixel (PCF).

2.2 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 centered at c with radius r, i.e., $\mathcal{B}_r(c) = \{x \in \mathbb{R}^n | ||c - x||_\infty \leq r\}$.

Aircraft Dynamical Model. In this chapter, we will consider an aircraft landing on a runaway. We assume the states of the aircraft to be measured with respect to the origin of the Runaway Coordinate Frame (shown in Figure 2.1(left)), where positions are: ξ_x is the axis across runaway; ξ_y is the altitude, and ξ_z is the axis along runaway. 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.

Runaway Parameters. We consider runaway 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 Runaway Coordinate Frame) i.e. $L = [(L_x, 0, L_z), (L_x + r_w, 0, L_z + r_l)]$ and $R = [(R_x, 0, R_z), (R_x + r_w, 0, R_z + r_l)]$ where r_w and r_l refers to the runaway width and length (standard international runaways are designed with $r_w = 40$ meters wide and $r_l = 3000$ meters.

Camera Model. We assume the aircraft is equipped with a monochrome camera $\mathcal{C}$ that produces images 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 runaway. 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 runaway parameters, for ease of notation, we drop this dependence from the notation $\mathcal{C}$.

We utilize an ideal pinhole camera model [32] to capture the image formation process of this camera. In general, a point p in the Runaway Coordinate Frame (`RCF`) is mapped into a point p' on the Camera Coordinate Frame (`CCF`) using a translation and rotation transformations defined by [20]:

$$\begin{bmatrix} p'_{x_{\mathtt{CCF}}} \\ p'_{y_{\mathtt{CCF}}} \\ p'_{z_{\mathtt{CCF}}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & \cos\theta & \sin\theta & y \\ 0 & -\sin\theta & \cos\theta & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{2.1}$$

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:

$$p'' = (p''_{x_{\mathtt{PCF}}}, p''_{y_{\mathtt{PCF}}}) = \left(\left\lfloor \frac{q_{x_{\mathtt{PCF}}}}{q_{z_{\mathtt{PCF}}}} \right\rfloor, \left\lfloor \frac{q_{y_{\mathtt{PCF}}}}{q_{z_{\mathtt{PCF}}}} \right\rfloor \right) \tag{2.2}$$

where:

$$\begin{bmatrix} q_{x_{\texttt{PCF}}} \\ q_{y_{\texttt{PCF}}} \\ q_{z_{\texttt{PCF}}} \end{bmatrix} = \begin{bmatrix} \rho_w & 0 & u_0 \\ 0 & -\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p'_{x_{\texttt{CCF}}} \\ p'_{y_{\texttt{CCF}}} \\ p'_{z_{\texttt{CCF}}} \\ 1 \end{bmatrix} \tag{2.3}$$

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 $u_0 = 0.5 * (\text{WP})$,$v_0 = 0.5 * (\text{HP})$, $\rho_w = \frac{\text{WP}}{\text{W}}$, $\rho_h = \frac{\text{HP}}{\text{H}}$. The values of each pixel in the final image I can be computed directly by checking if the point p'' lies within the area of the pixel and assigning 1 to such pixels accordingly [20].

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:

$$\text{visible} = \begin{cases} \text{yes} & |p''_{x_{\texttt{PCF}}}| \leq \frac{W}{2} \ \vee \ |p''_{y_{\texttt{PCF}}}| \leq \frac{H}{2} \\ \text{no} & \text{otherwise} \end{cases} \tag{2.4}$$

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.:

$$I[i, j] = \begin{cases} 1 & (p''_{x_{\texttt{PCF}}} == i - 1) \wedge (p''_{y_{\texttt{PCF}}} == j - 1) \wedge \text{visible} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

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 Runaway Coordinate Frame (`RCF`) to a pixel in the image I is summarized in Figure 2.1 (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., $\mathcal{NN} : \mathbb{B}^{q\times q} \to \mathbb{R}^m$. 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.,

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

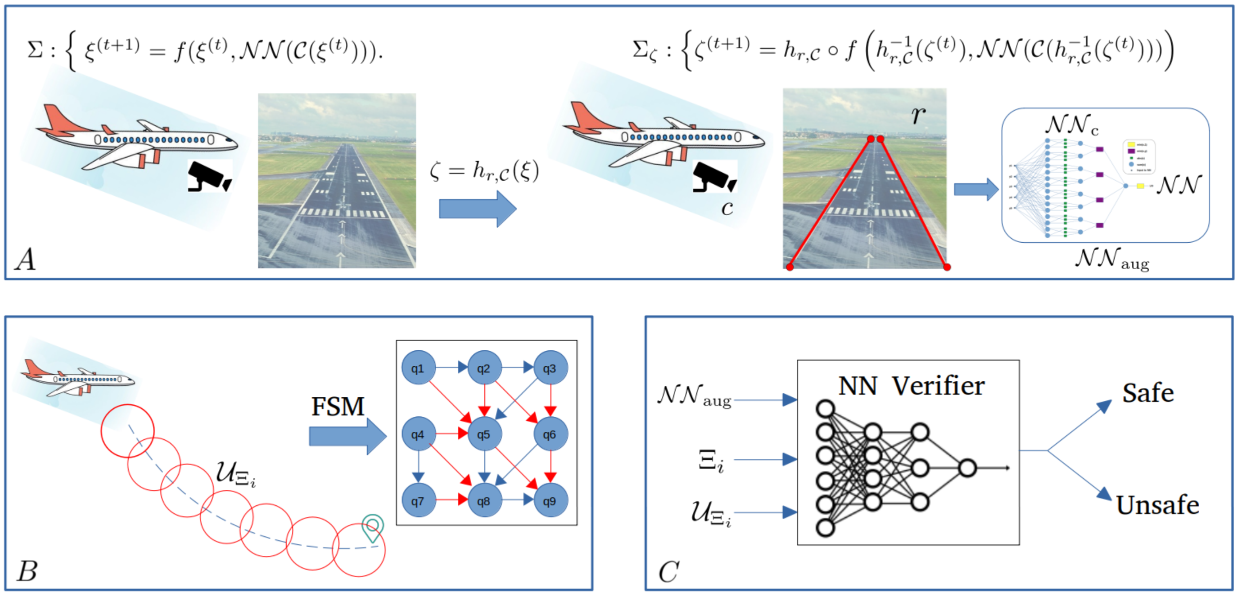
We are interested in checking if the closed-loop system meets some specifications that are captured using Linear Temporal Logic (LTL) (or a Bounded-Time LTL) formula φ. Examples of such formulas may include, but are not limited to:

- $\varphi_1 := \Diamond\{\xi_\theta = 0 \wedge \xi_y = 0\}$ 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 runaway (assuming a runaway length that is equal to 3000 meters).

For the formal definition of LTL and Bounded-Time LTL formulas syntax and semantics, we refer the reader to [6]. Given a formula φ that specifies safe 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 $\Sigma^{\mathcal{X}_0} \models \varphi$).

2.3 Framework

The verification problem described in Section 2.2 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 $\mathcal{NN}$.



$\Sigma : \left\{ \xi^{(t+1)} = f(\xi^{(t)}, \mathcal{NN}(\mathcal{C}(\xi^{(t)}))). \right.$
$\Sigma_\zeta : \left\{ \zeta^{(t+1)} = h_{r,\mathcal{C}} \circ f\left(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)}), \mathcal{NN}(\mathcal{C}(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)})))\right) \right.$
$\zeta = h_{r,\mathcal{C}}(\xi)$
r
c
$\mathcal{NN}_c$
$\mathcal{NN}$
$\mathcal{NN}_{\text{aug}}$
A
FSM
$\mathcal{U}_{\Xi_i}$
q1
q2
q3
q4
q5
q6
q7
q8
q9
B
$\mathcal{NN}_{\text{aug}}$
Ξ_i
$\mathcal{U}_{\Xi_i}$
NN Verifier
Safe
Unsafe
C

Figure 2.2: 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 vision 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 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:

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

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:

$$\forall \zeta \in \Xi. \big(\mathcal{NN}_{\text{aug}}(\zeta) \in \mathcal{U}_\Xi\big) \tag{2.6}$$

which can be easily verified using existing neural network model checkers [26, 31, 12]. 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 parameterized 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 parameterized each set $\mathcal{U}_{\Xi_i}$ as $\mathcal{U}_{\Xi_i} = \{u \in \mathbb{R}^4 | \; \|u - c_i\| \leq \mu_i\}$ where c_i is the control action taken at the center of Ξ_i and μ_i is the maximum ball around c_i such that the control actions $\mathcal{U}_{\Xi_i}$ guarantees satisfaction of the property φ. 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 Figure 2.2, 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 parameter μ_i and hence the set $\mathcal{U}_{\Xi_i}$.

- **(C) Apply the neural network model checker:** We use the neural network model checkers to verify that $\mathcal{NN}_{\text{aug}}$ satisfy (2.6) for each identified pair $(\Xi, \mathcal{U}_{\Xi_i})$.

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

2.4 Neural Network Augmentation

In this section, we focus on the problem of using the geometry of the runaway to develop a different mathematical model for the camera $\mathcal{C}$. As argued in the previous section and shown in Figure 2.3, our goal is to obtain a model with the same structure as 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 such a model $\mathcal{NN}_{\mathcal{C}}$ is the fact that ReLU based neural networks can only represent piece-wise affine (or linear) functions [34]. 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.

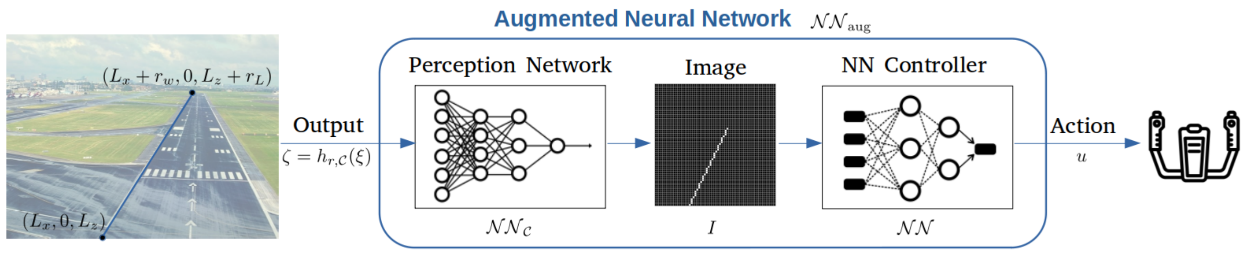

Augmented Neural Network $\mathcal{NN}_{\text{aug}}$
$(L_x + r_w, 0, L_z + r_L)$
$(L_x, 0, L_z)$
Output
$\zeta = h_{r,\mathcal{C}}(\xi)$
Perception Network
$\mathcal{NN}_{\mathcal{C}}$
Image
I
NN Controller
$\mathcal{NN}$
Action
u

Figure 2.3: Augmented network $\mathcal{NN}_{\text{aug}}$ maps the output ζ to control action u.

Change of Coordinates: Recall the runaway consists of line segments L and R (defined in Section 2.2). Instead of measuring the state of the aircraft by the 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:

$$\zeta = h_{r,\mathcal{C}}(\xi) = \begin{bmatrix} \zeta_1 \\ \\ \zeta_2 \\ \\ \zeta_3 \\ \\ \zeta_4 \\ \\ \zeta_5 \end{bmatrix} = \begin{bmatrix} \rho_w f \frac{L_x + \xi_x}{L_z cos(\xi_\theta) + \xi_z} + u_0 \\ \\ -\rho_h f \frac{L_z sin(\theta) + \xi_y}{L_z cos(\xi_\theta) + \xi_z} + v_0 \\ \\ \rho_w f \frac{(L_x + r_w) + \xi_x}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + u_0 \\ \\ -\rho_h f \frac{(L_z + r_L) sin(\xi_\theta) + \xi_y}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + v_0 \\ \\ \zeta_1 \zeta_4 - \zeta_2 \zeta_3 \end{bmatrix} \tag{2.7}$$

where $f, \rho_h, \rho_w, v_0, u_0$ are the camera physical parameters as defined in Section 2.2 . In other words, the pair (ζ_1, ζ_2) is the projection of the start point of the runaway $(L_x, 0, L_z)$ 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 runaway $(L_x + r_w, 0, L_z + r_L)$ onto the `PCF` frame. Indeed, we can define a similar set of variables for the other line segment of the runaway, R. The dependence of this change of coordinates on the camera parameters (e.g., the focal length f) and the runaway parameters (the length and width of the runaway) 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 2.1. *The change of coordinates function $h_{r,\mathcal{C}}$ is bijective.*

The proof of such proposition is based on ensuring that the inverse function $h_{r,\mathcal{C}}^{-1}$ 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:

$$\Sigma_\zeta : \begin{cases} \zeta^{(t+1)} = h_{r,\mathcal{C}} \circ f\left(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)}), \mathcal{NN}(\mathcal{C}(h_{r,\mathcal{C}}^{-1}(\zeta^{(t)})))\right) \end{cases} \tag{2.8}$$

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 as captured by the following proposition:

Proposition 2.2. *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 $\Xi_0 = \{h_{r,\mathcal{C}}(\xi) | \xi \in \mathcal{X}_0\}$.

Neural Network-based Model for Perception: While the model of the pinhole camera (defined in equation (2.1)-(2.5)) focuses on mapping individual points into pixels, we aim here to obtain a model that maps the entire runaway 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 runaway 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 equation (2.1)-(2.4)) 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 (2.2) and the logical checks in (2.4)-(2.5). 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 Figure 2.4 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 [7] which compute four values named O_1, O_2, O_3, O_4 as:

$$O_1 = \zeta_1(A_y - B_y) + \zeta_2(B_x - A_x) + A_x B_y - A_y B_x \tag{2.9}$$

$$O_2 = \zeta_3(A_y - B_y) + \zeta_4(B_x - A_x) + A_x B_y - A_y B_x \tag{2.10}$$

$$O_3 = -\zeta_1(A_y) + \zeta_2(A_x) + \zeta_3(A_y) - \zeta_4(A_x) + \zeta_5 \tag{2.11}$$

$$O_4 = -\zeta_1(B_y) + \zeta_2(B_x) + \zeta_3(B_y) - \zeta_4(B_x) + \zeta_5 \tag{2.12}$$

The line segment algorithm [7] detects an intersection whenever the following condition holds:

$$(\text{sign}(O_1) \neq \text{sign}(O_2)) \;\wedge\; (\text{sign}(O_3) \neq \text{sign}(O_4)) \tag{2.13}$$

Luckily, we can organize the equations (2.9)-(2.13) 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 (2.9)-(2.12) to assign the weights to the input layer of the neural network (as shown in Figure 2.4). To check the signs of $O_1, \ldots, O_4$, we recall the well-known identity for numbers of the same sign:

$$\text{sign}(a) = \text{sign}(b) \iff |a + b| - |a| - |b| = 0 \tag{2.14}$$

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

$$|x| = \max\{x, 0\} + \max\{-x, 0\}. \tag{2.15}$$

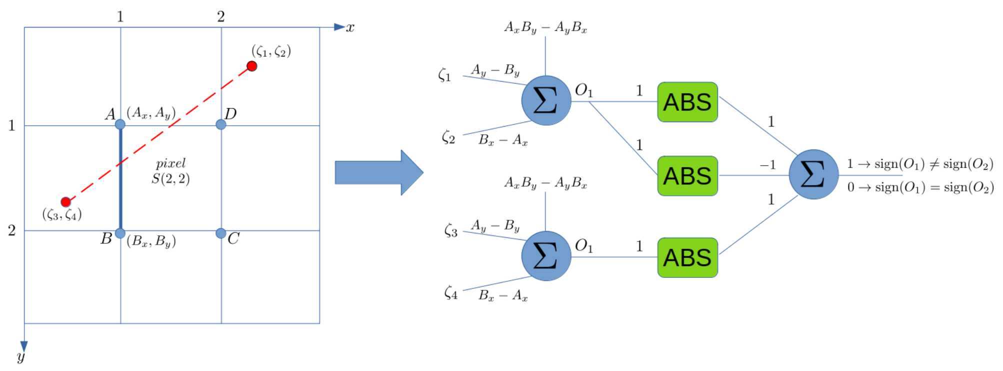


Figure 2.4: Line-segment intersection algorithm: The runaway line (in red) as seen by the camera intersects the pixel edge $A - B$ (in blue), this single edge intersection is detected by using a layer of six ReLU's.

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

$$min\{a, b\} = \frac{a+b}{2} - \frac{|a-b|}{2}. \tag{2.16}$$

The overall neural network requires $68 \times q \times q$ ReLU neurons for each projected line segment. The final architecture is shown in Figure 2.5. 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.,

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

Finally, by substituting in (2.8), we can now re-write the closed-loop dynamics as:

$$\Sigma_\zeta : \left\{ \zeta^{(t+1)} \quad = g\left(h^{-1}_{r,\mathcal{C}}(\zeta^{(t)}), \mathcal{NN}_{\text{aug}}(\zeta^{(t)})\right) \right. \tag{2.17}$$

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

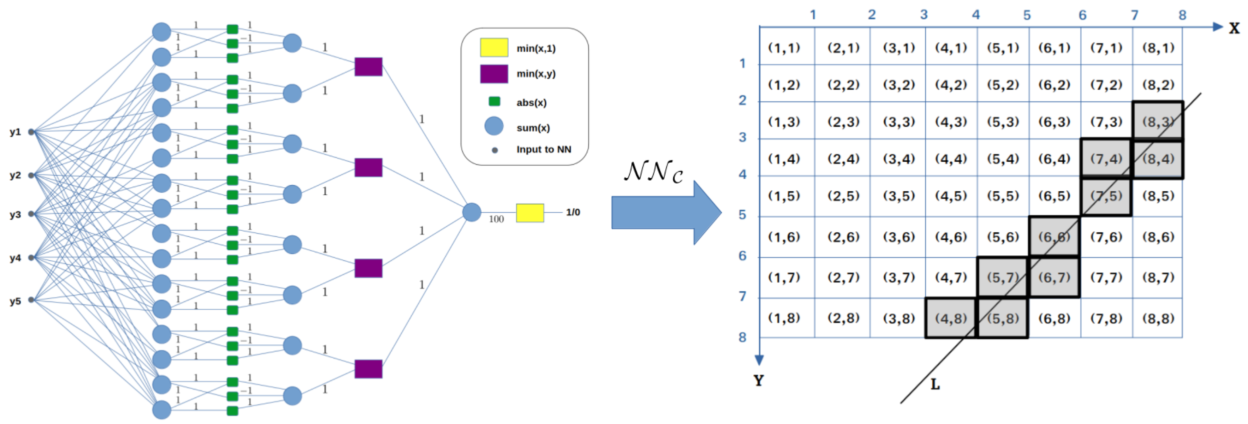


Figure 2.5: $\mathcal{NN}_{\mathcal{C}}$ checks the intersection between line segment $(\zeta_1, \zeta_2) - (\zeta_3, \zeta_4)$ and all edges of a each cell pixel of the final image.

2.5 Identifying the allowable control actions using symbolic abstractions

As shown in Section 2.3, our framework aims to split the verification of the dynamical system (2.17) 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 create 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 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 [23].

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 (2.17). Such finite state abstraction will take 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:

$$S_q = \{1, 2, \ldots L\} \quad \text{and} \quad j \in \sigma_q(i) \iff g\left(h^{-1}_{r,\mathcal{C}}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right) \in \Xi_j. \tag{2.18}$$

In other words, the finite state machine has a number of states L that is equal to the number of regions Ξ_i, i.e., each finite state symbolically represent 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 $g\left(h^{-1}_{r,\mathcal{C}}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right)$ 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 $_i \in \Xi_i$, 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) [47]. Given a center of the region c_i and an arbitrary state $\zeta_i \in \Xi_i$, the δ-FC assumption bounds the distance between the trajectories that starts at ζ_i and the center c_i, denoted by δ_ζ as:

$$\delta_\zeta \leq \beta(\epsilon, \tau) + \gamma(||\mathcal{NN}_{\text{aug}}(c_i) - \mathcal{NN}_{\text{aug}}(\zeta_i)||_\infty, \tau) \tag{2.19}$$

where τ is the sample time used to obtain the dynamics f (as explained in Section 2.2) and β and γ are class K_∞ functions that can be computed from the knowledge of the dynamics f. Such δ-FC 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 [46]. Given the inequality (2.19), we can revisit the definition of the state transition map σ_q to account for all possible trajectories as:

$$j \in \sigma_q(i) \Longleftrightarrow g\left(h_{r,\mathcal{C}}^{-1}(c_i), \mathcal{NN}_{\text{aug}}(c_i)\right) + \delta_\zeta \in \Xi_j. \tag{2.20}$$

With such a modification, one is direct to show that following result:

Proposition 2.3. *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:*

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

where $\mathcal{S}_I = \{i \in \{1, \ldots, L\} | \exists \zeta_0 \in \Xi_0, \zeta_i \in \Xi_i\}$.

This proposition follows directly from Theorem 4.1 in [47].

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 [24], 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 (2.19) in a "backward design approach". We first search for the maximum value of δ_ζ that renders Σ_q compatible with the specification. For 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:

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

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

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

1  $\{\Xi_1, \Xi_2, \ldots, \Xi_L\}$ = Partition_into_regions($\Xi, \epsilon$)
2  $\mu = \underline{\mu}$
3  statusFSM $\leftarrow$ UNSAT
4  while statusFSM == UNSAT do
5      $\Sigma_q$ = Create_FSM($f, h, h^{-1}, \tau, \beta, \gamma, \mathcal{NN}_{\text{aug}}, \{\Xi_1, \ldots, \Xi_L\}, \mu$)
6      statusFSM = Check_FSM ($\varphi$, $\Sigma_q$, $T$)
7      if $\mu \leq \overline{\mu}$ then
8          $\mu$ = Increase_MU($\mu$)
9      end
10 end
11 for $i = 1$ to $L$ do
12     STATUS_NN[$i$] = NN_Verifier ($\mathcal{NN}_{\text{aug}}$, $\Xi_i$, $\mu$)
13     if STATUS_NN[i] == SAT then
14         STATUS = UNSAFE
15         return STATUS
16     end
17 end
18 STATUS = SAFE
19 return STATUS
```
Algorithm 1: LanderNN-VeriF

2.6 Numerical Example

We illustrate the results in this chapter on a Vision-Based Aircraft landing system. For a fixed-wing aircraft defined using the guidance kinematic model [4], where orientations (in Rads) are defined by the course angle χ (rotation around $y_{\texttt{CCF}}$ axis), pitch angle θ (rotation around $x_{\texttt{CCF}}$ 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 runaway ($\chi = 0$), similarly velocity is kept as constant. Moreover, $\dot{\theta}$ (Rad/s) is regarded as control input u.

The state vector of the aircraft at time $t \in \mathbb{N}$ is denoted by $\xi^{(t)} = [\xi_\theta^{(t)}, \xi_x^{(t)}, \xi_y^{(t)}, \xi_z^{(t)}]^T$ and is assumed to evolve over time while being governed by the dynamical system [4]

$$\xi_z^{(t+1)} = \xi_z^{(t)} + V_g \tau \cos(\xi_\theta^{(t)}) \tag{2.21}$$

$$\xi_y^{(t+1)} = \xi_y^{(t)} + V_g \tau \sin(\xi_\theta^{(t)}) \tag{2.22}$$

$$\xi_\theta^{(t+1)} = \xi_\theta^{(t)} + u^{(t)} \tau \tag{2.23}$$

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 Runaway 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 (2.21)-(2.23) is a δ-FC system. In particular, by using the method [46] and the δ-FC Lyapunov function $\mathcal{V}(\xi^{(t)}, \xi^{(t)\prime}) = (\xi^{(t)} - \xi^{(t)\prime})^2$ one can show:

$$\beta(\zeta_1, \zeta_2, \zeta_3, \tau) = \sqrt{8}\,\sqrt{\zeta_1^2 + \zeta_2^2 + \zeta_3^2}\; e^{\tau} \tag{2.24}$$

$$\gamma(\mu, \tau) = \sqrt{V_g(e^{2\tau} - 1)}\mu \tag{2.25}$$

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 $16 \times 16 \times 16$ 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 Figure 2.6 and its corresponding camera view is shown in Figure 2.7.

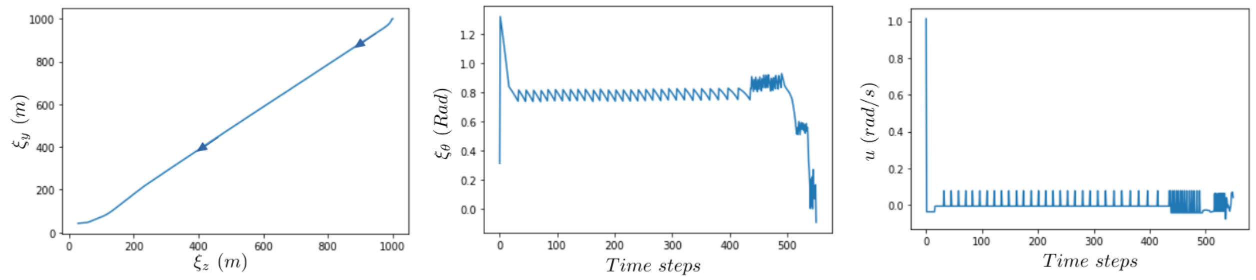


Figure 2.6: Aircraft landing using augmented controller $\mathcal{NN}_{aug}$. Left: aircraft position (ξ_y, ξ_z); Middle: aircraft angle (ξ_θ); Right: aircraft control $(u = \mathcal{NN}_{aug})$.

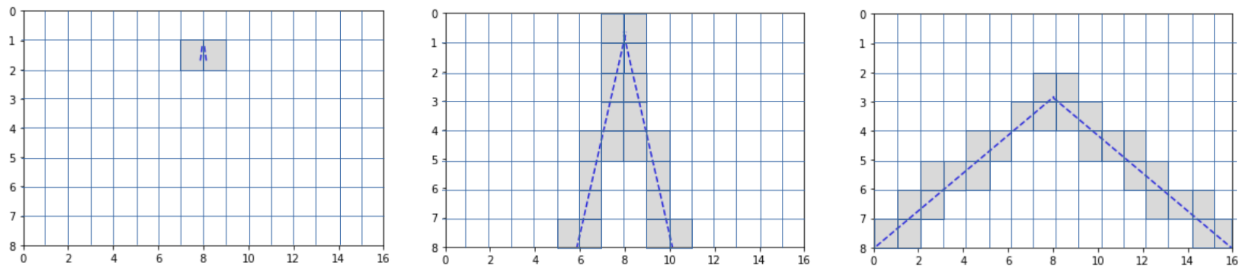


Figure 2.7: Landing camera view using 16×16 pixels resolution. Left: $\xi^1 = [1000, 1000, \frac{\pi}{4}]$, Middle: $\xi^{300} = [400, 300, \frac{\pi}{8}]$, Right: $\xi^{1000} = [5, 5, 0]$.

We used a Boolean SAT solver named SAT4J [5] 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 μ 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 the PeregriNN [31] as the NN model checker. Figure 2.8 reports the execution time for verifying the neural network property in 100 random regions, and Figure 2.9 in regions 1 to 500. The average execution time was 75 seconds and the NN was found to be safe and satisfying the specification φ.

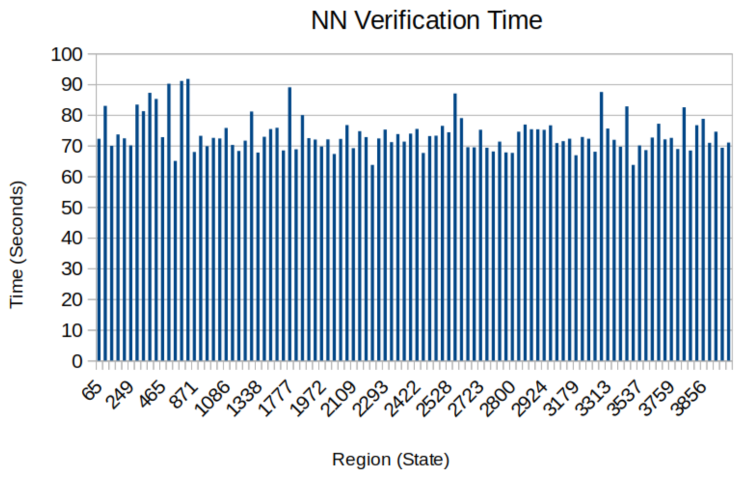


Figure 2.8: Execution time for verifying the neural network property in 100 random regions.

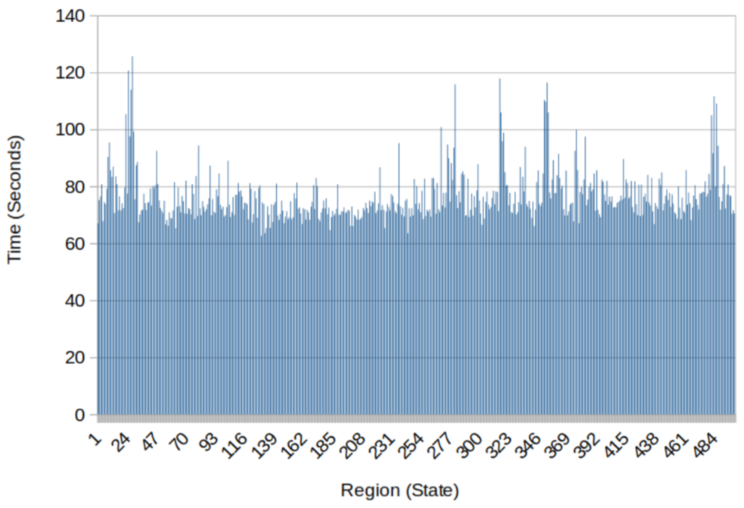


Figure 2.9: Execution time for verifying the neural network property in regions 1 to 500.

Part II

Training of Probably Correct Perception Based AI Models

Chapter 3

Certified Vision-based State Estimation for Autonomous Landing Systems using Reachability Analysis

This chapter studies the problem of designing a certified vision-based state estimator for autonomous landing systems. In such a system, a neural network (NN) processes images from a camera to estimate the aircraft's relative position with respect to the runway. We propose an algorithm to design such NNs with certified properties in terms of their ability to detect runways and provide accurate state estimation. At the heart of our approach is the use of geometric models of perspective cameras to obtain a mathematical model that captures the relation between the aircraft states and the inputs. We show that such geometric models enjoy mixed monotonicity properties that can be used to design state estimators with certifiable error bounds. We show the effectiveness of the proposed approach using an experimental testbed on data collected from event-based cameras.

3.1 Introduction

Machine learning models, like deep neural networks, are increasingly used to control dynamical systems in safety-critical applications. These black-box models trained using data are used heavily to process high-dimensional imaging data like LiDAR scanners and cameras to produce state estimates to low-level, model-based controllers. While these deep Neural Networks (NNs) provide empirically accepted results, they lack certified guarantees in terms of their ability to process complex scenes and provide estimates of the location of different objects within the scene. It is then unsurprising the increasing number of reported failures of these deep NNs in building reliable autonomous systems.

In this chapter, we will consider the safety of deep neural networks that control aircraft while approaching runways to perform an autonomous landing. Such a problem enjoys geometric nature that can be exploited to develop a geometrical/physical model of the perception system. Yet, it represents a significant real-world problem of interest to the designers of the autonomous system. In particular, we present a novel neural network-based filter that can process complex scenes along with estimates of the state of the aircraft—computed by unverified complex deep neural networks—and output a state estimate of the aircraft with a certified error bound. That is, akin to the "control shields" in the reinforcement learning literature [11, 2], the proposed filter can be thought of as a "shield" that can filter out incorrect estimates of the aircraft and replaces them with ones with certified error bounds. In contrast, the correct estimates pass this filter (or shield) unaltered.

A central challenge to designing such a filter is the need to explicitly model the imaging process, i.e., the relation between the system state and the images created by the camera [35]. An early result on the application of formal verification for vision-based dynamical systems controlled with neural networks [39] focused only on the usage of LiDARs.

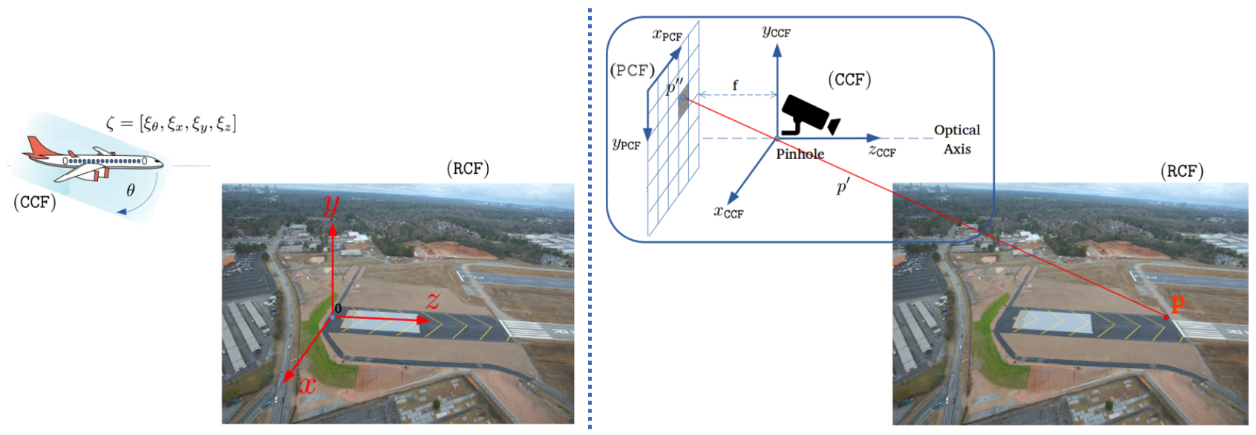


Figure 3.1: Coordinate frames: Runway (RCF), Camera (CCF) and Pixel (PCF).

The first steps in formally modeling the imaging process for camera-based systems have been recently studied in [22, 28, 19]. In particular, the work in [22, 28] proposes the use of abstractions of the perception system as a formal model of perception. Unfortunately, these abstractions are only tested on a set of samples and lack guarantees in their ability to model the perception system formally. The work in [19] extends the notion of imaging-adapted partitions, originally defined for LiDAR images [39], to the notion of image-invariant regions, which are regions within which the captured images are identical. Unfortunately, the work in [19] focuses only on simple scenes that can be modeled as a collection of triangles that represent the triangulated faces of objects in the environment. The work in [41] considers the problem of estimating the pose of different objects in the scene. Given a partial point cloud of an object, the goal is to estimate the object's pose and provide a certificate of correctness for the resulting estimate. While capable of handling complex objects, the framework in [41] is sound but not complete, meaning that if it can identify the object's pose, it will generate a certificate. Still, not all poses of the object will be identified, even if the object of interest exists in the scene. Other techniques include classification that uses targeted inputs with the aim of finding counterexamples that violate safety [36]. However, such techniques do not provide formal guarantees regarding the ability to find all counterexamples.

In this chapter, we build on our recent results [35] that exploit 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 physics-based formal model is designed such that it can be encoded as a neural network (with manually chosen weights) that we refer to as the Runway Generative Model neural network. Thanks to the recent development in computing the reachable sets of neural networks (the set of all possible outputs of the network) [44, 16, 42], we can characterize the set of all possible images for the runway. We use such reachability analysis to design novel filters that can remove all the other objects in the scene by matching the spatial and geometrical properties of the runway to those in the computed reachable set. Moreover, as a by-product of this design, the proposed filter identifies the set of possible state estimates of the aircraft. This set of possible state estimates can then be used to cross-check the ones computed by unverified neural network estimators and provide certifiable error bounds on the final state estimate.

3.2 Preliminaries

3.2.1 Notation

We 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 $x \in \mathbb{R}^n$. We denote by $\mathcal{B}_r(c)$ the infinity norm centered at c with radius r, i.e., $\mathcal{B}_r(c) = \{x \in \mathbb{R}^n | ||c - x||_\infty \leq r\}$. We use the notation $A_{[i,j]}$ to denote the element in the i^{th} row and j^{th} column of A. Analogously, the notation $A_{[i,:]}$ denotes the i^{th} row of A, and $A_{[:,j]}$ denotes the j^{th} column of A; when A is a vector instead, both notations return a scalar. Let $\mathbf{0}_{n,m}$ be an $(n \times m)$ matrix of zeros, and $\mathbf{1}_{n,m}$ be the $(n \times m)$ matrix of ones. Finally, the symbols $\oplus$ and $\otimes$ denote element-wise addition and multiplication of matrices.

3.2.2 Aircraft State Space

In this paper, we 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 Figure 3.1 (left)), where positions are: ξ_x is the axis across runway; ξ_y is the altitude and ξ_z is the axis along the runway. We consider only one angle ξ_θ, representing the pitch rotation around the x axis of the aircraft. The state vector of the aircraft at time $t \in \mathbb{N}$ is denoted by $\xi^{(t)} = [\xi_\theta^{(t)}, \xi_x^{(t)}, \xi_y^{(t)}, \xi_z^{(t)}]^T$.

3.2.3 Runway Parameters

We consider a runway that consists of two border line segments, L and R. Each line segment can be characterized by its start and end point (also measured in the Runway Coordinate Frame) i.e., $L = [(L_x, 0, L_z), (L_x + r_w, 0, L_z + r_l)]$ and $R = [(R_x, 0, R_z), (R_x + r_w, 0, R_z + r_l)]$ where r_w and r_l refers to the runway width and length (e.g. standard international runways are designed with $r_w = 40$ meters wide and $r_l = 3000$ meters).

3.2.4 Camera Model

We assume the aircraft is equipped with a monochrome camera $\mathcal{C}$ that produces images of $a \times b$ 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 and the other objects in the scene. In other words, we can model the camera $\mathcal{C}$ as a function that maps aircraft states into images, i.e., $\mathcal{C} : \mathbb{R}^4 \to \mathbb{B}^{a \times b}$. Although the images created by the camera depend on the runway parameters and the other objects in the scene, we drop this dependence from the notation $\mathcal{C}$ for ease of notation.

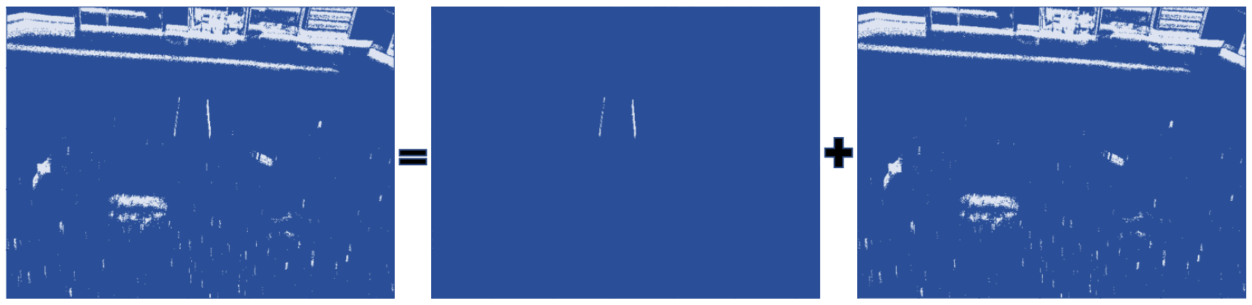


Figure 3.2: Monochromatic images generated using state-of-the-art event-based cameras. The full image I to the left can be decomposed into one that contains only the runway image I_r (center) and the remaining objects/noise I_n (right), i.e., $I = I_r + I_n$.

We utilize an ideal pinhole camera model [32] to capture the image formation process of this camera. In general, a point p in the Runway Coordinate Frame (RCF) is mapped into a point p' on the Camera Coordinate Frame (CCF) using a translation and rotation transformations defined by [20]:

$$\begin{bmatrix} p'_{x_{\texttt{CCF}}} \\ p'_{y_{\texttt{CCF}}} \\ p'_{z_{\texttt{CCF}}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & \cos\theta & \sin\theta & y \\ 0 & -\sin\theta & \cos\theta & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{3.1}$$

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:

$$p'' = (p''_{x_{\texttt{PCF}}}, p''_{y_{\texttt{PCF}}}) = \left(\left\lfloor \frac{q_{x_{\texttt{PCF}}}}{q_{z_{\texttt{PCF}}}} \right\rfloor, \left\lfloor \frac{q_{y_{\texttt{PCF}}}}{q_{z_{\texttt{PCF}}}} \right\rfloor \right) \tag{3.2}$$

where:

$$\begin{bmatrix} q_{x_{\texttt{PCF}}} \\ q_{y_{\texttt{PCF}}} \\ q_{z_{\texttt{PCF}}} \end{bmatrix} = \begin{bmatrix} \rho_w & 0 & u_0 \\ 0 & -\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p'_{x_{\texttt{CCF}}} \\ p'_{y_{\texttt{CCF}}} \\ p'_{z_{\texttt{CCF}}} \\ 1 \end{bmatrix} \tag{3.3}$$

and f is the focal length of the camera lens, W is the image width (in meters), H is the image height (in meters), a is the image Width (in pixels), b is the image height (in pixels), and $u_0 = 0.5a$, $v_0 = 0.5b$, $\rho_w = \frac{\mathrm{a}}{\mathrm{W}}$, $\rho_h = \frac{\mathrm{b}}{\mathrm{H}}$. The values of each pixel in the final image I can be computed directly by checking if the point p'' lies within the area of the pixel and assigning 1 to such pixels accordingly [20].

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

$$\text{visible} = \begin{cases} \text{yes} & |p''_{x_{\mathtt{PCF}}}| \leq \frac{W}{2} \ \vee \ |p''_{y_{\mathtt{PCF}}}| \leq \frac{H}{2} \\ \text{no} & \text{otherwise} \end{cases} \tag{3.4}$$

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

$$I_{[k,l]} = \begin{cases} 1 & (p''_{x_{\mathtt{PCF}}} == k-1) \wedge (p''_{y_{\mathtt{PCF}}} == l-1) \wedge \text{visible} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

for $k \in (1, 2, 3...\text{a})$ and $l \in (1, 2, 3...\text{b})$. This process of mapping a point p in the Runway Coordinate Frame (`RCF`) to a pixel in the image I is summarized in Figure 3.1 (right).

Since the scene contains both a runway and other unknown objects (see Figure 3.2), we define the final image $I \in \mathbb{B}^{a \times b}$ captured by the camera as:

$$I(\xi) = I_r(\xi) + I_n(\xi) \tag{3.6}$$

where $I_r \in \mathbb{B}^{a \times b}$ is the image corresponding to the existence of the runway in the scene and $I_n \in \mathbb{B}^{a \times b}$ is the image corresponding to the existence of other objects/noise in the scene.

3.2.5 Neural Network Estimator

We are interested in designing a Neural Network (NN)-based estimator that can process an image $I(\xi) = I_r(\xi) + I_n(\xi)$ to produce an estimate of the aircraft state ξ. An F-layer NN is specified by composing F layer functions (or just layers). A layer ω with $\mathfrak{i}_\omega$ inputs and $\mathfrak{o}_\omega$ outputs is specified by a weight matrix $W^\omega \in \mathbb{R}^{\mathfrak{o}_\omega \times \mathfrak{i}_\omega}$ and a bias vector $b^\omega \in \mathbb{R}^{\mathfrak{o}_\omega}$ as follows:

$$L_{\theta^\omega} : z \mapsto \phi(W^\omega z + b^\omega), \tag{3.7}$$

where ϕ is a nonlinear function, and $\theta^\omega \triangleq (W^\omega, b^\omega)$ for brevity. Thus, an F-layer NN is specified by F layer functions $\{L_{\theta^\omega} : \omega = 1, \ldots, F\}$ whose input and output dimensions are composable: that is, they satisfy $\mathfrak{i}_\omega = \mathfrak{o}_{\omega-1}$, $\omega = 2, \ldots, F$. Specifically:

$$\mathcal{NN}(I) = (L_{\theta^F} \circ L_{\theta^{F-1}} \circ \cdots \circ L_{\theta^1})(I). \tag{3.8}$$

As a common practice, we allow the output layer L_{θ^F} to omit the nonlinear function ϕ.

3.2.6 Problem Formulation

Problem 3.1. *Given an image $I(\xi) = I_r(\xi) + I_n(\xi)$ that contains the projection of a runway and other unknown objects and an estimation error $\epsilon > 0$, design a neural network estimator $\mathcal{NN}$ such that $||\mathcal{NN}(I_r + I_n) - \xi|| < \epsilon$.*

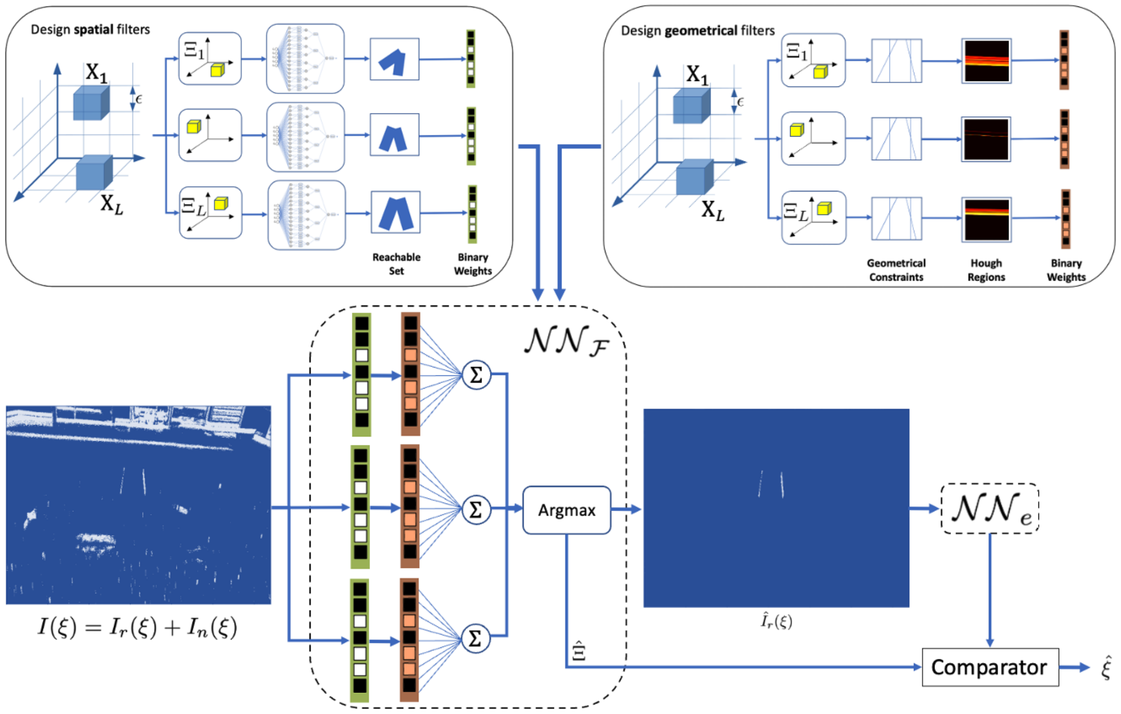


Figure 3.3: Overall framework elements: Spatial filter, Geometrical filter, $\mathcal{NN}_\mathcal{F}$ and $\mathcal{NN}_e$

3.3 Framework

Classical machine learning approaches to solve Problem 3.1 entail training neural networks on large labeled data sets that contain different possibilities of runway positions and surrounding objects. Since ensuring the correctness of the resulting NN is challenging, we propose a framework in which we manually design a NN filter $\mathcal{NN}_\mathcal{F}$ that is guaranteed to "filter out" the noise I_n, i.e., $\mathcal{NN}_\mathcal{F}(I_r + I_n) = I_r$. Moreover, such a filter $\mathcal{NN}_\mathcal{F}$ also computes a certified bound on the possible states of the aircraft $\hat{\Xi}$. The size of this possible set of states $\hat{\Xi}$ is chosen to guarantee the ϵ bound in Problem 3.1. The resulting filtered-out image $\mathcal{NN}_\mathcal{F}(I_r + I_n)$ is then passed into a neural network estimator $\mathcal{NN}_e$ that is trained using existing techniques in machine learning.

Finally, the outcome of $\mathcal{NN}_e$ is checked against the certified bounds $\hat{\Xi}$ to provide the final estimate as:

$$\hat{\xi} = \begin{cases} \mathcal{NN}_e\left(\mathcal{NN}_{\mathcal{F}}(I)\right) & \text{if } \mathcal{NN}_e\left(\mathcal{NN}_{\mathcal{F}}(I)\right) \in \hat{\Xi} \\ \text{center}(\hat{\Xi}) & \text{otherwise} \end{cases} \tag{3.9}$$

where center($\hat{\Xi}$) is well defined whenever the set $\hat{\Xi}$ is a hypercube. In other words, the certified bounds $\hat{\Xi}$ are used to *replace* the incorrect state estimates with ones with guaranteed error bound from within the set $\hat{\Xi}$. This process is depicted in Figure 3.3. Steps to manually design the NN filter $\mathcal{NN}_{\mathcal{F}}$ and its theoretical guarantees are given in the subsequent subsections.

3.3.1 Physics-based Generative Model for Runway Images

Our prior work in [35] developed a physics-based generative model that can generate all possible images containing runways $I_r(\xi)$ based on the physical parameters of the camera $f, \rho_h, \rho_w, v_0, u_0$ (discussed in Section 3.2). Crucially, this physics-based generative model was shown to be mathematically equal to a change of coordinates $h : \mathbb{R}^4 \to \mathbb{R}^4$ and a neural network $\mathcal{NN}_r(h(\xi))$ with carefully selected weights and parameters (this network is depicted in Figure 3.4), i.e.,

$$I_r(\xi) = \mathcal{NN}_r(h(\xi)).$$

The change of coordinates h maps the state of the aircraft into the projections of the end-points of the lines L and R on the Pixel Coordinate Frame (PCF).

The generative model is made of two components: A change of coordinates h and a ReLU-based Neural Network for Line Generation $\mathcal{NN}_r$ as captured by the following definitions.

Definition 3.1 (Change of Coordinates)**.** *We define the change of coordinates as:*

$$\zeta = h(\xi) = \begin{bmatrix} \zeta_1 \\ \\ \zeta_2 \\ \\ \zeta_3 \\ \\ \zeta_4 \end{bmatrix} = \begin{bmatrix} \rho_w f \frac{L_x + \xi_x}{L_z cos(\xi_\theta) + \xi_z} + u_0 \\ \\ -\rho_h f \frac{L_z sin(\theta) + \xi_y}{L_z cos(\xi_\theta) + \xi_z} + v_0 \\ \\ \rho_w f \frac{(L_x + r_w) + \xi_x}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + u_0 \\ \\ -\rho_h f \frac{(L_z + r_L) sin(\xi_\theta) + \xi_y}{(L_z + r_L) cos(\xi_\theta) + \xi_z} + v_0 \end{bmatrix} \tag{3.10}$$

where $f, \rho_h, \rho_w, v_0, u_0$ are the camera physical parameters as defined in Section 3.2.

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 $(L_x + r_w, 0, L_z + r_L)$ onto the `PCF` frame. Indeed, we can define a similar set of variables for the other line segment of the runway, R. We refer to the new state space as Ξ.

Definition 3.2 (Runway Generative Model)**.** *We define the Runway Generative Model $\mathcal{NN}_r$ of $a \times b$ pixels as:*

$$I_r(\xi) = \mathcal{NN}_r(h(\xi)) \tag{3.11}$$

for simplicity let's consider $\zeta = h(\xi)$, then:

$$\mathcal{NN}_r(\zeta) = ReLU(\phi_{Min}(\phi_{Abs}(\phi_{Lin}(\zeta, \phi_{Det}(\zeta))))) \tag{3.12}$$

where:

$$\zeta_{det} = \phi_{Det}(\zeta) = \zeta_1\zeta_4 - \zeta_2\zeta_3 \tag{3.13}$$

$$\zeta^1 = \phi_{Lin}(\zeta, \zeta_{det}) = W[\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_{det}]^T \tag{3.14}$$

$$\zeta^2 = \phi_{Abs}(\zeta^1) = \begin{bmatrix} -|\zeta^1_{[1]} + \zeta^1_{[2]}| + |\zeta^1_{[1]}| + |\zeta^1_{[2]}| \\ \vdots \\ -|\zeta^1_{[15]} + \zeta^1_{[16]}| + |\zeta^1_{[15]}| + |\zeta^1_{[16]}| \end{bmatrix} \tag{3.15}$$

$$\zeta^3 = \phi_{Min}(\zeta^2) = \begin{bmatrix} \min(\zeta^2_{[1]}, \zeta^2_{[2]}) \\ \vdots \\ \min(\zeta^2_{[7]}, \zeta^2_{[8]}) \end{bmatrix} \tag{3.16}$$

where $\phi_{Det} : \mathbb{R}^4 \to \mathbb{R}$, $\phi_{Lin} : \mathbb{R}^5 \to \mathbb{R}^{16\times q}$, $\phi_{Abs} : \mathbb{R}^{16\times q} \to \mathbb{R}^{8\times q}$, $\phi_{Min} : \mathbb{R}^{8\times q} \to \mathbb{R}^{4\times q}$ *and* $ReLU : \mathbb{R}^{4\times q} \to \mathbb{R}^q$, *where* $q = a \times b$ *is the number of pixels in the image to be generated, weight matrix* $W \in \mathbb{R}^{16\times 5}$ *contains fixed weights fully decribed by camera parameters [35]. We refer to* $I_r(\xi)$ *as the image containing solely the runway defined by* ζ.

We refer the reader to [35] for detailed analysis of the correctness of this generative model.

3.3.2 Design of spatial filters using output reachability analysis

Given a partitioning parameter δ, we partition the state space $\Xi \subset \mathbb{R}^4$ into L regions $\Xi_1, \ldots, \Xi_L$ such that each Ξ_i is an infinity-norm ball with radius δ. For each of these partitions, we aim to design a spatial filter that matches the spatial properties of the runway images that can be produced by states within such a partition. To that end, consider the following filter $\mathcal{S}^{\Xi_i} \in \mathbb{B}^{a\times b}$ defined as:

$$\mathcal{S}^{\Xi_i} = \bigotimes_{h(\xi)\in\Xi_i} I_r(\xi) = \bigotimes_{h(\xi)\in\Xi_i} \mathcal{NN}_r(h(\xi)). \tag{3.17}$$

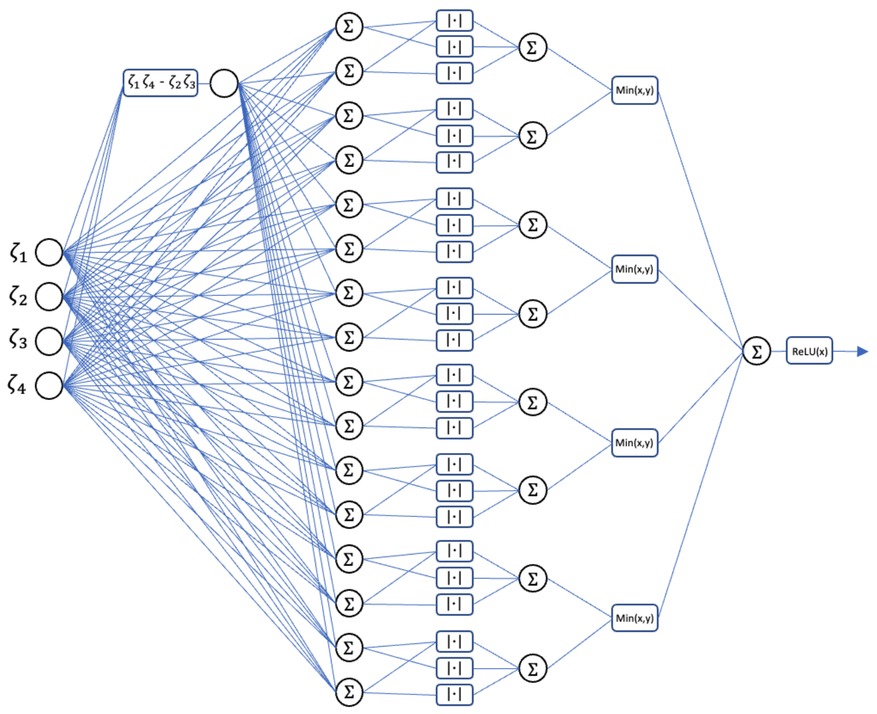


Figure 3.4: Physics-based generative model for runway images $I_r(\xi)$ captured mathematically as a neural network [35].

Recall that all images $I_r(\xi)$ are monochromatic (i.e., each pixel can take only a value of 0 or 1), then the following result follows directly from the definition above.

Proposition 3.1. *Consider the filter $\mathcal{S}^{\Xi_i}$. The following holds:*

$$(i)\xi \in \Xi_i,\ \forall \xi \in \Xi_i.[I_n(\xi) \otimes \mathcal{NN}_r(h(\xi)) = \mathbf{0}_{a,b}]$$

$$\implies [I_r(\xi) + I_n(\xi)] \otimes \mathcal{S}^{\Xi_i} = I_r(\xi) \tag{3.18}$$

$$(ii)\xi \notin \Xi_i,\ I_n(\xi) \notin \mathcal{I}_r^{\Xi_i}$$

$$\implies [I_r(\xi) + I_n(\xi)] \otimes \mathcal{S}^{\Xi_i} \neq I_r(\xi) \tag{3.19}$$

where $\mathcal{I}_r^{\Xi_i} = \{I_r(\xi) \in \mathbb{B}^{a \times b} | h(\xi) \in \Xi_i\}$.

Note that the condition $\forall \xi \in \Xi_i.[I_n(\xi) \otimes \mathcal{NN}_r(h(\xi)) = \mathbf{0}_{a,b}]$ is equivalent to $I_n(\xi) \otimes \mathcal{S}^{\Xi^i} = \mathbf{0}_{a,b}$. That is, the filter $\mathcal{S}^{\Xi_i}$ is capable of removing all noise in the image as long as the noise image $I_n(\xi)$ does not affect pixels that are δ/ρ_w away from the runway image $I_r(\xi)$.

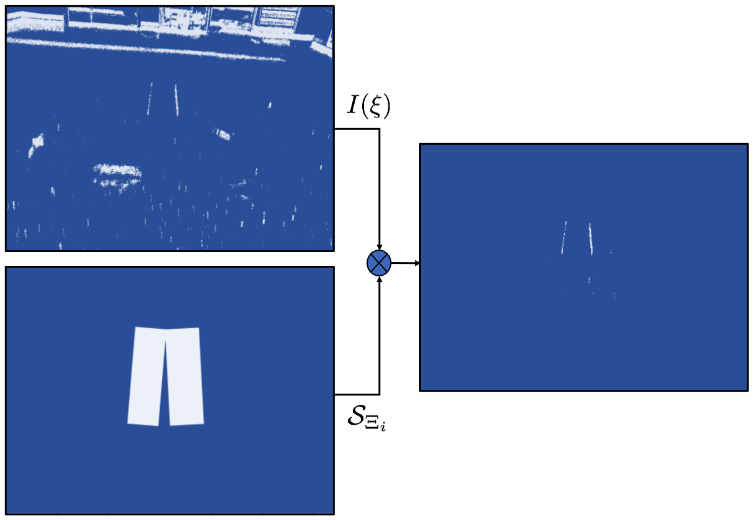


Figure 3.5: Spatial filtering focuses attention on different regions.

Figure 3.5 shows an example of such a filter. Specifically, equations (3.18)-(3.19) imply that the filter will accurately process the filtered image, provided that the noise does not resemble the pattern of runways. Additionally, the filter must be applied to the specific region corresponding to the state responsible for generating such a runway. Furthermore, it is reasonable to assume that as we increase the geometric complexity of the runway, the likelihood of noise resembling runway patterns diminishes. In other words, the more intricate the entity we are examining, the safer it is to rely on our assumptions.

What is remaining is to provide an algorithm that can compute the filter $\mathcal{S}^{\Xi_i}$ for each partition Ξ_i. Thanks to the fact that the physics-based generative model $\mathcal{NN}_r(\xi)$ is captured as a neural network, one can use output reachability algorithms to compute an overapproximation of the reach set (set of all possible images) for the runway image $I_r(\xi)$. For that end, we leverage Mixed-monotonicity reachability analysis of neural networks [33] by leveraging the following result:

Proposition 3.2. *(from [33]) Given a neural network $\mathcal{NN} : \mathbb{R}^{\mathfrak{i}} \to \mathbb{R}^{\mathfrak{o}}$ and an interval $[\underline{J}, \overline{J}] \subseteq \mathbb{R}^{\mathfrak{o}\times\mathfrak{i}}$ bounding the derivative of $\mathcal{NN}$ for all input $\zeta \in [\underline{\zeta}, \overline{\zeta}]$. Let us denote the center of the interval as J^* and for each output dimension $i \in \{1, ..., \mathfrak{o}\}$, define input vectors $\underline{\zeta}_{[i,:]}, \overline{\zeta}_{[i,:]} \in \mathbb{R}^{\mathfrak{i}}$ and a row vector $\alpha^i \in \mathbb{R}^{1\times\mathfrak{i}}$ such that for all $j \in \{1, ..., \mathfrak{i}\}$ the following holds:*

$$(\underline{\psi}_{[i,j]}, \overline{\psi}_{[i,j]}, \alpha_{[i,j]}) = \begin{cases} (\underline{\zeta}_{[:,j]}, \overline{\zeta}_{[:,j]}, min(0, \underline{J}_{[i,j]})) & \textit{if } J^*_{[i,j]} \geq 0 \\ (\overline{\zeta}_{[:,j]}, \underline{\zeta}_{[:,j]}, max(0, \overline{J}_{[i,j]})) & \textit{if } J^*_{[i,j]} \leq 0 \end{cases} \tag{3.20}$$

Then for all neural network input $\zeta \in [\underline{\zeta}, \overline{\zeta}]$ *and* $i \in \{1, ..., \mathfrak{o}\}$*, we have:*

$$\begin{aligned} \mathcal{NN}(\zeta)_{[i,:]} \in [&\mathcal{NN}(\underline{\psi}_{[i,:]} - \alpha_{[i,:]}(\underline{\psi}_{[i,:]} - \overline{\psi}_{[i,:]})), \\ &\mathcal{NN}(\overline{\psi}_{[i,:]} + \alpha_{[i,:]}(\underline{\psi}_{[i,:]} - \overline{\psi}_{[i,:]}))] \end{aligned} \tag{3.21}$$

To implement the method in Proposition 3.2, we define the input vectors as $\overline{\zeta} = \text{center}(\Xi_i) + \frac{\delta}{2}$ and $\underline{\zeta} = \text{center}(\Xi_i) - \frac{\delta}{2}$. Additionally, we compute the bounds on the Jacobian matrix of the neural network $\mathcal{NN}_r$ to find the bounds $[\underline{J}, \overline{J}]$.

We compute the bound on the Jacobian matrix of the Runway Generative Model as follows:

$$J_{\phi_{Det}} = \begin{bmatrix} \zeta_4 \\ -\zeta_3 \\ -\zeta_2 \\ \zeta_1 \end{bmatrix} \tag{3.22}$$

$$J_{\phi_{Lin}} = \begin{bmatrix} W_{[1,1]} & W_{[2,1]} & \dots \\ \vdots & \ddots & \\ W_{[1,5]} & & W_{[16,5]} \end{bmatrix} \tag{3.23}$$

$J_{\phi_{Abs}}$ bounds are:

$$\underline{J}_{\phi_{Abs}} = \begin{bmatrix} -2 & 0 & \dots \\ -2 & 0 & \dots \\ 0 & -2 & \dots \\ 0 & -2 & \dots \\ \vdots & \ddots & \\ 0 & & -2 \\ 0 & & -2 \end{bmatrix} \; ; \; \overline{J}_{\phi_{Abs}} = \begin{bmatrix} 2 & 0 & \dots \\ 2 & 0 & \dots \\ 0 & 2 & \dots \\ 0 & 2 & \dots \\ \vdots & \ddots & \\ 0 & & 2 \\ 0 & & 2 \end{bmatrix} \tag{3.24}$$

Such bounds are well defined, provided that all inputs $\zeta_1, \zeta_2, \zeta_3, \zeta_4 > 0$, which is the case for all practical applications.

$J_{\phi_{Min}}$ bounds are:

$$\underline{J}_{\phi_{Min}} = \mathbf{0}_{8,4} \; ; \; \overline{J}_{\phi_{Min}} = \mathbf{1}_{8,4} \tag{3.25}$$

Such bounds come from the fact that:

$$\frac{\partial(Min(\zeta_i, \zeta_j))}{\partial \zeta_i} = \begin{cases} 0 & \zeta_i \leq \zeta_j \\ 1 & \text{otherwise} \end{cases} \tag{3.26}$$

These bounds on the output of $\mathcal{NN}_r$ identifies which pixels are equal to zero for all the images generated by the states in each Ξ_i, which can be used to compute the filters in (3.17).

3.3.3 Design of Geometric Filters using Hough Transform

The spatial filters $\mathcal{S}^{\Xi_i}$ can be used to focus the attention on different regions of the state space. Although these filters provide a guarantee of the output of the filter that satisfies $\xi \in \Xi_i$ it does not provide any guarantee on the output of the filters for which $\xi \notin \Xi_i$. Therefore, it is necessary to augment the spatial filters with another filter that aims to detect whether the output follows the geometrical structure of the runway images.

To achieve this, consider the following filters:

$$\mathcal{H}^{\Xi_i}(I) = \begin{cases} 1 & \text{if } \exists \xi \in \Xi_i \text{ such that } I = \mathcal{NN}_r(h(\xi)) \\ 0 & \text{otherwise} \end{cases} \tag{3.27}$$

Such filter can be efficiently computed using the classical Hough-space transformation [40]. In this transformation, a straight line is represented by a normal line that passes through the origin and is orthogonal to that straight line. The equation of the normal line is given by $\rho = \zeta_1 \cos(\theta) + \zeta_2 \sin\theta$, where ρ is the length of the normal line and θ is the angle between the normal line and the x-axis of the Pixel Coordinate Frame. By using the projections of the endpoints of the runway lines edges obtained from $h(\xi) = [\zeta_1, \zeta_2, \zeta_3, \zeta_4]$ as $P_1 = (\zeta_1, \zeta_2)$ and $P_2 = (\zeta_3, \zeta_4)$, we can solve for θ and ρ for the generated image as:

$$\theta = \tan^{-1}\left(\frac{\zeta_1 - \zeta_3}{\zeta_4 - \zeta_2}\right) \tag{3.28}$$

$$\rho = \zeta_1 \cos(\theta) + \zeta_2 \sin(\theta) \tag{3.29}$$

Given a partition Ξ_i, we can obtain the range of ρ, θ for all runway images as follows. First, recall that each partition Ξ_i is an infinity ball with a radius equal to δ around a center point $\text{center}(\Xi_i) \in \mathbb{R}^4$. The two points $P_1 = (\text{center}(\Xi_i)_{[1]}, \text{center}(\Xi_i)_{[2]})$ and $P_2 = (\text{center}(\Xi_i)_{[3]}, \text{center}(\Xi_i)_{[4]})$ represent 2-dimensional points in the Pixel Coordinate

Frame that corresponds to the center of Ξ_i (see Figure 3.6 for illustration). Following the 2-dimensional geometry of the Pixel Coordinate Frame, it is direct to show that:

$$(\zeta_1^{c_i}, \zeta_2^{c_i}, \zeta_3^{c_i}, \zeta_4^{c_i}) = \text{center}(\Xi_i) \tag{3.30}$$

$$\theta_{max}^{\Xi_i} = \begin{cases} \tan^{-1}\left(\frac{\zeta_1^{c_i}-\zeta_3^{c_i}+2\delta}{\zeta_4^{c_i}-\zeta_2^{c_i}+2\delta}\right), & \text{if } \frac{\zeta_4^{c_i}-\zeta_2^{c_i}}{\zeta_3^{c_i}-\zeta_1^{c_i}} > 0 \\ \tan^{-1}\left(\frac{\zeta_1^{c_i}-\zeta_3^{c_i}+2\delta}{\zeta_4^{c_i}-\zeta_2^{c_i}-2\delta}\right), & \text{otherwise} \end{cases} \tag{3.31}$$

$$\theta_{min}^{\Xi_i} = \begin{cases} \tan^{-1}\left(\frac{\zeta_1^{c_i}-\zeta_3^{c_i}-2\delta}{\zeta_4^{c_i}-\zeta_2^{c_i}-2\delta}\right), & \text{if } \frac{\zeta_4^{c_i}-\zeta_2^{c_i}}{\zeta_3^{c_i}-\zeta_1^{c_i}} > 0 \\ \tan^{-1}\left(\frac{\zeta_1^{c_i}-\zeta_3^{c_i}-2\delta}{\zeta_4^{c_i}-\zeta_2^{c_i}+2\delta}\right), & \text{otherwise} \end{cases} \tag{3.32}$$

$$\rho_{min}^{\Xi_i} = \underline{b}_\delta \frac{\sqrt{1+m^2}}{m+\frac{1}{m}} \tag{3.33}$$

$$\rho_{max}^{\Xi_i} = \overline{b}_\delta \frac{\sqrt{1+m^2}}{m+\frac{1}{m}} \tag{3.34}$$

where $m, \overline{b}_\delta, \underline{b}_\delta$ are defined as

$$m = \frac{\zeta_4^{c_i} - \zeta_2^{c_i}}{\zeta_3^{c_i} - \zeta_1^{c_i}} \tag{3.35}$$

$$\overline{b}_\delta = (\zeta_2^{c_i} + \delta) - m(\zeta_1^{c_i} + \delta) \tag{3.36}$$

$$\underline{b}_\delta = (\zeta_2^{c_i} - \delta) - m(\zeta_1^{c_i} - \delta) \tag{3.37}$$

Equations (3.30)-(3.34) define the reachable set of the runway images within the Hough space (the $\rho-\theta$ space). Moreover, the discretization introduced in the Pixel Coordinate Frame (the flooring operation in (3.2)) introduces a discretization over the range of ρ and θ computed by equations (3.30)-(3.34) which existing implementations of Hough transformation algorithms take into account. We denote by $\mathcal{L}^{\Xi_i} = \{(\rho_{max}^{\Xi_i}, \theta_{\max}^{\Xi_i}), \ldots, (\rho_{min}^{\Xi_i}, \theta_{min}^{\Xi_i})\}$ the discrete set of the allowable values of ρ and θ within the partition Ξ_i.

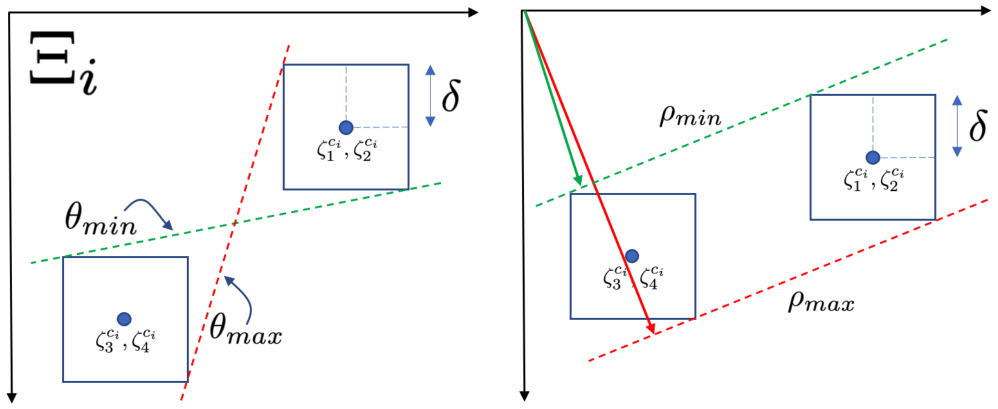

Ξ_i
δ
$\zeta_1^{c_i}, \zeta_2^{c_i}$
θ_{min}
$\zeta_3^{c_i}, \zeta_4^{c_i}$
θ_{max}
ρ_{min}
δ
$\zeta_1^{c_i}, \zeta_2^{c_i}$
$\zeta_3^{c_i}, \zeta_4^{c_i}$
ρ_{max}

Figure 3.6: Feasible range of angles and distances in Hough Space.

For each possible (ρ_j, θ_j) in $\mathcal{L}^{\Xi_i}$, we define the filter $\mathcal{R}^{\Xi_i}(\rho_j, \theta_j) \in \mathbb{B}^{a \times b}$ as:

$$\mathcal{R}^{\Xi_i}(\rho_j, \theta_j)_{[k,l]} = \begin{cases} 1 & \text{if } l - 1 < -\frac{\cos\theta_j}{\sin\theta_j}k + \frac{\rho_j}{\sin\theta_j} < l \\ 0 & \text{otherwise} \end{cases} \tag{3.38}$$

For each filter $\mathcal{R}^{\Xi_i}(\rho_j, \theta_j)$ we can define a mismatching score that computes how far the input image I is from the expected output of this filter as:

$$\mathcal{M}(I, \mathcal{R}^{\Xi_i}(\rho_j, \theta_j)) = \left\| - I \oplus \mathcal{R}^{\Xi_i}(\rho_j, \theta_j) \right\|_1 \tag{3.39}$$

That is, $\mathcal{M}$ is equal to zero whenever the input image I matches exactly the line represented by $\mathcal{R}^{\Xi_i}(\rho_j, \theta_j)$ and non-zero otherwise. Finally, we can implement the filter $\mathcal{H}^{\Xi_i}$ in (3.27) as:

$$\mathcal{H}^{\Xi_i}(I) = \begin{cases} 1 & \text{if } \arg\min\{\mathcal{M}(I, \mathcal{R}^{\Xi_i}(\rho^{\Xi_i}_{max}, \theta^{\Xi_i}_{max})), \dots, \\ & \qquad \mathcal{M}(I, \mathcal{R}^{\Xi_i}(\rho^{\Xi_i}_{min}, \theta^{\Xi_i}_{min}))\} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.40}$$

In other words, the filter $\mathcal{H}^{\Xi_i}$ produces 1 whenever any of the filters $\mathcal{R}^{\Xi_i}(\rho^{\Xi_i}_{max}, \theta^{\Xi_i}_{max}), \dots, \mathcal{R}^{\Xi_i}(\rho^{\Xi_i}_{min}, \theta^{\Xi_i}_{min})$ were able to match its input image. The following proposition follows directly from the definition of $\mathcal{H}^{\Xi_i}(I)$ above.

Proposition 3.3. *Consider the filter $\mathcal{H}^{\Xi_i}$ defined in* (3.40). *The following holds:*

$$\mathcal{H}^{\Xi_i}(I) = 1 \Longleftrightarrow \exists \xi \in \Xi_i \text{ such that } I = \mathcal{NN}_r(h(\xi)) \tag{3.41}$$

3.3.4 Design of the NN filter $\mathcal{NN}_{\mathcal{F}}$

The final filter $\mathcal{NN}_F$ consists of processing the images I using all the spatial filters $\mathcal{S}^{\Xi_1}, \dots, \mathcal{S}^{\Xi_l}$ followed by the geometric filters $\mathcal{H}^{\Xi_1}, \dots, \mathcal{H}^{\Xi_l}$. Finally, the filter $\mathcal{NN}_F$ identifies the partition $\hat{\Xi}$ for which the geometric filter returns 1 to produce its final outputs as follows:

$$\hat{\Xi} = \arg\max\{\mathcal{H}^{\Xi_1}(I \otimes \mathcal{S}^{\Xi_1}), \dots, \mathcal{H}^{\Xi_l}(I \otimes \mathcal{S}^{\Xi_l})\} \tag{3.42}$$

$$\hat{I}_r = I \otimes \mathcal{S}^{\hat{\Xi}} \tag{3.43}$$

The following result captures the correctness of the $\mathcal{NN}_F$.

Theorem 3.1. *Consider a noisy image $I(\xi) = I_r(\xi) + I_n(\xi)$, a partitioning of the state space Ξ into infinity balls of radius δ namely $\Xi_1, \dots, \Xi_l$. Denote by Ξ^* the partition for which the aircraft state ξ belongs, i.e., $h(\xi) \in \Xi^*$. Under the following assumptions:*

$$(i) I_n(\xi) \notin \{\mathcal{NN}_r(h(\xi)) \mid h(\xi) \in \Xi\} \tag{3.44}$$

$$(ii) \forall \xi \in \Xi^*. [I_n(\xi) \otimes \mathcal{NN}_r(h(\xi)) = \mathbf{0}_{a,b}] \tag{3.45}$$

then the following holds:

$$\hat{\Xi} = \Xi^* \tag{3.46}$$

$$\hat{I}_r = I_r(\xi) \tag{3.47}$$

$$\|\xi - \hat{\xi}\| \leq 4L_h\delta \qquad \forall \hat{\xi} \in \hat{\Xi} \tag{3.48}$$

where $(\hat{\Xi}, \hat{I}_r) = \mathcal{NN}_F(I(\xi))$ and L_h is the Lipschitz constant of h^{-1}.

Proof. We start by proving (3.46) as follows. For the sake of contradiction, we assume that there exists a partition $\Xi^\dagger \neq \hat{\Xi}$ such that which the aircraft state ξ satisfies $h(\xi) \in \Xi^\dagger$. It follows from Proposition 3.1 and assumptions (3.44) and (3.45) that:

$$I \otimes \mathcal{S}^{\hat{\Xi}} \neq I_r(\xi), \qquad I \otimes \mathcal{S}^{\Xi^\dagger} = I_r(\xi)$$

and hence Proposition 3.3 entails that:

$$\mathcal{H}^{\hat{\Xi}}(I \otimes \mathcal{S}^{\hat{\Xi}}) = 0, \qquad \mathcal{H}^{\Xi^\dagger}(I \otimes \mathcal{S}^{\Xi^\dagger}) = 1$$

Nevertheless, this contradicts the fact that:

$$\hat{\Xi} = \arg\max\{\ldots, \mathcal{H}^{\hat{\Xi}}(I \otimes \mathcal{S}^{\hat{\Xi}}), \ldots, \mathcal{H}^{\Xi^\dagger}(I \otimes \mathcal{S}^{\Xi^\dagger}), \ldots\}$$

which proves that $h(\xi) \in \hat{\Xi}$.

Equation (3.47) follows directly from (3.46) and Proposition 3.1. Similarly, equation (3.48) follows from the fact that the partition $\hat{\Xi}$ is an infinity ball of radius δ and hence for any $\hat{\xi} \in \hat{\Xi}$:

$$\begin{aligned}\|h(\xi) - h(\hat{\xi})\|_\infty &= \|h(\xi) + \text{center}(\hat{\Xi}) - \text{center}(\hat{\Xi}) - h(\hat{\xi})\|_\infty \\ &\leq \|h(\xi) - \text{center}(\hat{\Xi})\|_\infty + \|\text{center}(\hat{\Xi}) - h(\hat{\xi})\|_\infty \leq 2\delta\end{aligned}$$

Hence from the relation between the 2-norm and the infinity norm, we conclude that:

$$\|h(\xi) - h(\hat{\xi})\| \leq \sqrt{4}\|h(\xi) - h(\hat{\xi})\|_\infty \leq 4\delta$$

from which we conclude that:

$$\|\xi - \hat{\xi}\| \leq 4L_h\delta$$

which concludes the proof. □

Before we conclude this section, it is essential to interpret the assumptions (3.44) and (3.45) in Theorem 3.1. In particular, the assumption in (3.44) entails that the noise I_n can not be generated using the runway generative model $\mathcal{NN}_r$. In other words, this assumption ensures that the noise does not look like a runway and hence only one image of the runway exists in the scene. The assumption in (3.45) asks that the pixels that are δ close to the runway are not affected by the noise. It is crucial to note that assumption (3.45) is required to be satisfied in Ξ^* only and does not affect other partitions.

3.4 Experimental Evaluation

We present the results of a vision-based aircraft landing system that uses a target runway. We consider two runway segments, $L = [(L_x, 0, L_z), (L_x, 0, L_z+r_l)]$ and $R = [(R_x, 0, R_z), (R_x, 0, R_z+r_l)]$ where $R_x = 0.1$, $L_x = -0.1$, $R_z = 0$, $L_z = 0$, $r_l = 0.3$ (in meters).

To generate monochromatic images, we utilized the SilkyEvCam event-based camera with a resolution of 640×480 pixels, a focal length of 8 mm, and a pixel size of 15 $\mu m \times 15$ μm. We measured the ground-truth states of the vehicle using Vicon motion capture cameras to track optical markers attached to the camera envelope, and the centroid of the camera was defined as the camera coordinate frame (CCF) origin. Similarly, we defined the runway target as the runway coordinate frame (RCF) from which all measurements were made.

We partitioned the state space $\Xi \subset \mathbb{R}^4$ into 27 regions $\Xi_1, \ldots, \Xi_{27}$ using a partitioning parameter $\delta = 0.1$. These regions correspond to the range of states $[\xi_y \times \xi_z \times \xi_\theta] = [0.8, 1] \times [1.6, 1.8] \times [0.5, 0.7]$ (we fix $\xi_x = 0$ in our experiments). We then implemented the runway generative model neural network $\mathcal{NN}_r$ for a resolution of 640×480 pixel images, the filter $\mathcal{NN}_f$, and the corresponding application of the spatial $\mathcal{S}^{\Xi_i}$ and geometric filters $\mathcal{H}^{\Xi_i}$ on all partitions to create the binary weights needed using PyTorch libraries.

This process took approximately 20 minutes per partition, resulting in a total of approximately 9 hours to generate the neural network weights for all 27 partitions using an Apple M1 Pro processor with 32 GB of RAM.

Next, the filter $\mathcal{NN}_F$ was used to process images collected from the SilkyEvCam event-based camera. We operated the camera for several minutes resulting in a total of 1320 images using 25 frames per second. Figure 3.7 and Figure 3.8 show two instances of the images collected and processed during our experiments. As seen from the two figures, the scene contains one runway and several objects, and noisy pixels. The neural network $\mathcal{NN}_F$ is used to filter these images and remove all objects except for the runway. Figure 3.7(right) and Figure 3.8(right) show the outputs of the 4 different spatial filters $\mathcal{S}^{\Xi_i}$. As can be observed in the two figures, the result of these filters focuses the attention on specific segments of the scene. Some of these filtered images contain the runway (or segments of it) while others contain only parts of the noise image I_n. Next, we execute the geometric filters $\mathcal{H}^{\Xi_i}$ to identify the images that match the geometric structure of the runways. We highlight the partition with the smallest mismatch score $\mathcal{M}$ with a green box in Figure 3.7 and Figure 3.8. In particular, in Figure 3.7, the output corresponding to partition 1 contains leads to the smallest mismatch score while partition 24 corresponds to the one with the smallest mismatch score in Figure 3.8.

Finally, we used off-the-shelf algorithms to process the filtered image and produce the final state estimate. For the test reported in Figure 3.7, the resulting state error is 0.0777 while for the test reported in Figure 3.8 the resulting error is 0.045, both are below the threshold of $4\delta L_h$ and hence no further processing is required.

Additionally, for comparison purposes, we applied an off-the-shelf standard Hough transformation-based filter that can discover line segments in the scene with the aim to identify the runway without our proposed filter. Figure 3.9 shows the output of the standard Hough transformation-based filter when operated on the same input image used in Figure 3.7.

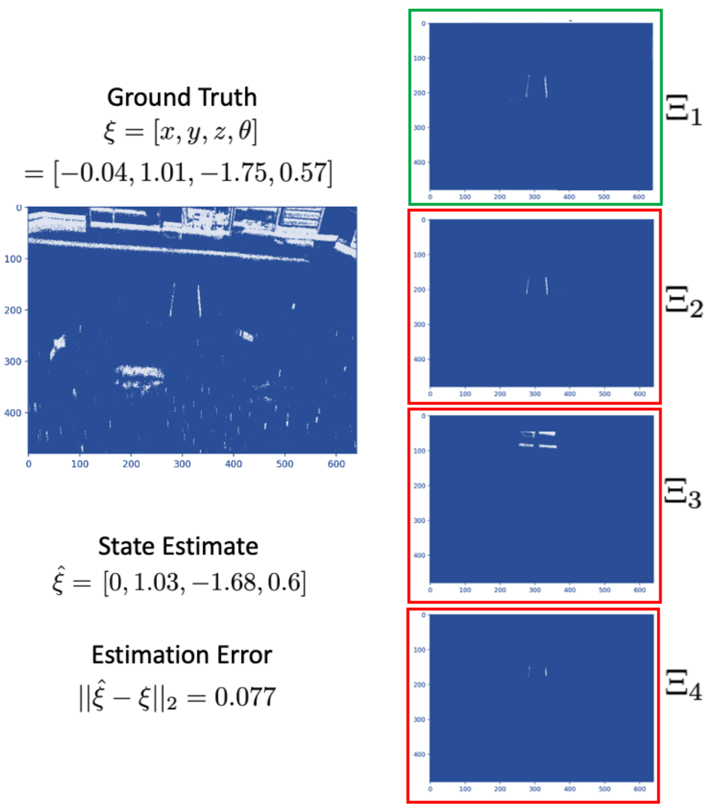


Figure 3.7: Test 1: Framework application on image #1 delivers correct filtered runway (in Green) found on Partition #1.

The dashed lines in Figure 3.9 correspond to the line segments that were detected by the standard filter. As can be appreciated from Figure 3.9, the standard filter leads to several false line detections that do not match the runway due to the noise and the other objects in the scene. Fortunately, our proposed filter does not suffer from such an issue and comes with provable guarantees.

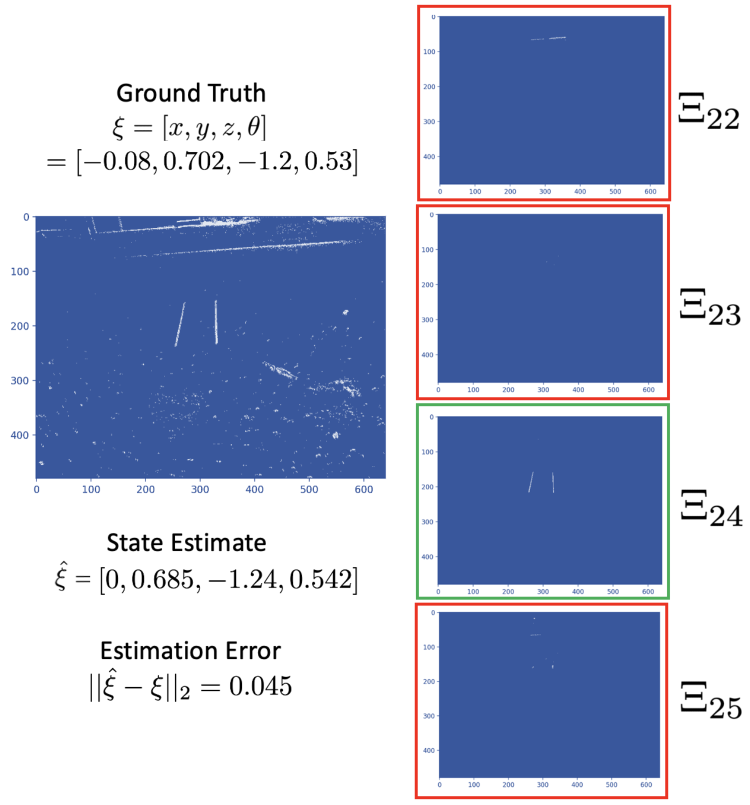


Figure 3.8: Test 2: Framework application on image #2 delivers correct filtered runway (in Green) found on Partition #24.

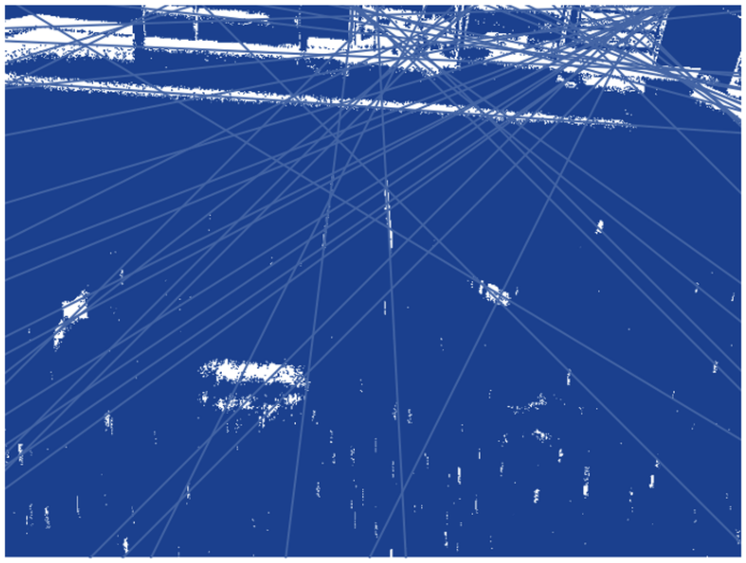


Figure 3.9: Filtering using only Hough filter without geometrical constraints.

Part III

Repair of Faulty AI Models

Chapter 4

Safe-by-Repair: A Convex Optimization Approach for Repairing Unsafe Two-Level Lattice Neural Network Controllers

In this chapter, we consider the problem of *repairing* a data-trained Rectified Linear Unit (ReLU) Neural Network (NN) controller for a discrete-time, input-affine system. That is we assume that such a NN controller is available, and we seek to repair unsafe closed-loop behavior at one known "counterexample" state while simultaneously preserving a notion of safe closed-loop behavior on a separate, verified set of states. To this end, we further assume that the NN controller has a Two-Level Lattice (TLL) architecture, and exhibit an algorithm that can systematically and efficiently repair such an network. Facilitated by this choice, our approach uses the unique semantics of the TLL architecture to divide the repair problem into two significantly decoupled sub-problems, one of which is concerned with repairing the un-safe counterexample – and hence is essentially of local scope – and the other of which ensures that the repairs are realized in the output of the network – and hence is essentially of global scope. We then show that one set of sufficient conditions for solving each these sub-problems can be cast as a convex feasibility problem, and this allows us to formulate the TLL repair problem as two separate, but significantly decoupled, convex optimization problems. Finally, we evaluate our algorithm on a TLL controller on a simple dynamical model of a four-wheel-car.

4.1 Introduction

The proliferation of Neural Networks (NNs) as safety-critical controllers has made obtaining provably correct NN controllers vitally important. However, most current techniques for doing so involve a repeatedly training and verifying a NN until adequate safety properties have been achieved. Such methods are not only inherently computationally expensive (because training and verification of NNs are), their convergence properties can be extremely poor. For example, when verifying multiple safety properties, such methods can cycle back and forth between safety properties, with each subsequent retraining achieving one safety property by undoing another one.

An alternative approach obtains safety-critical NN controllers by *repairing* an existing NN controller. Specifically, it is assumed that an already-trained NN controller is available that performs in a *mostly* correct fashion, albeit with some specific, known instances of incorrect behavior. But rather than using retraining techniques, repair entails *systematically* altering the parameters of the original controller *in a limited way*, so as to retain the original safe behavior while simultaneously correcting the unsafe behavior. The objective of repair is to exploit as much as possible the safety that was learned during the training of the original NN parameters, rather than allowing re-training to *unlearn* safe behavior.

Despite these advantages, the NN repair problem is challenging because it has two main objectives, both of which are at odds with each other. In particular, repairing an unsafe behavior requires altering the NN's response in a *local* region of the state space, but changing even a few neurons generally affects the *global* response of the NN – which could undo the initial safety guarantee supplied with the network. This tension is especially relevant for general deep NNs, and repairs realized on neurons in their latter layers. This is especially the case for repairing controllers, where the relationship between specific neurons and their importance to the overall safety properties is difficult to discern. As a result, there has been limited success in studying NN controller repair, especially for nonlinear systems.

In this chapter, we exhibit an explicit algorithm that can repair a NN controller for a *discrete-time, input-affine nonlinear* system. The cornerstone of our approach is to consider NN controllers of a specific architecture: in particular, the recently proposed Two-Level Lattice (TLL) NN architecture [13]. The TLL architecture has unique neuronal semantics, and those semantics greatly facilitate finding a balance between the local and global trade-offs inherent in NN repair. In particular, by assuming a TLL architecture, we can separate the problem of controller repair into two *significantly decoupled* problems, one consisting of essentially only local considerations and one consisting of essentially only global ones.

Related Work: Repairing (or patching) NNs can be traced to the late 2000s. An early result on patching connected transfer learning and concept drift with patching [29]; another result established fundamental requirements to apply classifier patching on NNs by using inner layers to learn a patch for concept drift in an image classifier network [30]. Another approach based on a Satisfiability Modulo Theory (SMT) formulation of the repair problem was proposed by [37] where they changed the parameters of a classifier network to comply with a safety specification, i.e. where the designer knows exactly the subset of the input space to be classified. This prior work nonetheless is heuristic-based and so not guaranteed to produced desired results, which was noticed by [18] who cast the problem of patching (minimal repair) as a verification problem for NNs (including Deep ones). However, this work focused on a restricted version of the problem in which the changes in weights are limited to a single layer. Finally, [8] proposed a verification-based approach for repairing DNNs but not restricted to modifying the output; instead, proposed to identify and modify the most relevant neurons that causes the safety violation using gradient guidance.

4.2 Preliminaries

4.2.1 Notation

We will denote the real numbers by $\mathbb{R}$. For an $(n \times m)$ matrix (or vector), A, we will use the notation $[\![A]\!]_{i,j}$ to denote the element in the i^{th} row and j^{th} column of A. Analogously, the notation $[\![A]\!]_{i,\cdot}$ will denote the i^{th} row of A, and $[\![A]\!]_{\cdot,j}$ will denote the j^{th} column of A; when A is a vector instead of a matrix, both notations will return a scalar corresponding to the corresponding element in the vector. Let $\mathbf{0}_{n,m}$ be an $(n \times m)$ matrix of zeros. We will use bold parenthesis $\boldsymbol{(}\cdot\boldsymbol{)}$ to delineate the arguments to a function that *returns a function.* We use the functions `First` and `Last` to return the first and last elements of an ordered list (or a vector in $\mathbb{R}^n$). The function `Concat` concatenates two ordered lists, or two vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$ along their (common) nontrivial dimension to get a third vector in $\mathbb{R}^{n+m}$. Finally, $B(x;\delta)$ denotes an open Euclidean ball centered at x with radius δ. The norm $\|\cdot\|$ will refer to the Euclidean norm.

4.2.2 Dynamical Model

In this chapter, we will consider the general case of a discrete-time input-affine nonlinear system Σ specified by:

$$\Sigma : \begin{cases} x_{i+1} = f(x_i) + g(x_i)u_i \end{cases} \tag{4.1}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the input. In addition, $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ are continuous and smooth functions of x.

Definition 4.1 (Closed-loop Trajectory)**.** *Let $u : \mathbb{R}^n \to \mathbb{R}^m$. Then a **closed-loop trajectory** of the system* (4.1) *under u, starting from state x_0, will be denoted by the sequence $\{\zeta_i^{x_0}(u)\}_{i=0}^{\infty}$. That is $\zeta_{i+1}^{x_0}(u) = f(\zeta_i^{x_0}(u)) + g(\zeta_i^{x_0}(u)) \cdot u(\zeta_i^{x_0}(u))$ and $\zeta_0^{x_0}(u) = x_0$.*

Definition 4.2 (Workspace)**.** *We will assume that trajectories of* (4.1) *are confined to a connected, compact workspace, X_{ws} with non-empty interior, of size ext$(X_{\mathsf{ws}}) \triangleq \sup_{x \in X_{\mathsf{ws}}} \|x\|$.*

4.2.3 Neural Networks

We will exclusively consider Rectified Linear Unit Neural Networks (ReLU NNs). A K-layer ReLU NN is specified by composing K *layer* functions, each of which may be either linear and nonlinear. A *nonlinear* layer with $\mathfrak{i}$ inputs and $\mathfrak{o}$ outputs is specified by a $(\mathfrak{o} \times \mathfrak{i})$ real-valued matrix of *weights*, W, and a $(\mathfrak{o} \times 1)$ real-valued matrix of *biases*, b as follows: $L_\theta : z \mapsto \max\{Wz + b, 0\}$ with the max function taken element-wise, and $\theta \triangleq (W, b)$. A *linear* layer is the same as a nonlinear layer, only it omits the nonlinearity $\max\{\cdot, 0\}$; such a layer will be indicated with a superscript *lin*, e.g. L_θ^{lin}. Thus, a K-layer ReLU NN function as above is specified by K layer functions $\{L_{\theta^{(i)}} : i = 1, \dots, K\}$ that are *composable*: i.e. they satisfy $\mathfrak{i}_i = \mathfrak{o}_{i-1} : i = 2, \dots, K$.

We will annotate a ReLU function $\mathscr{NN}$ by a *list of its parameters* $\Theta \triangleq (\theta^{|1}, \ldots, \theta^{|K})$[1]. The number of layers and the *dimensions* of the matrices $\theta^{|i} = (\ W^{|i}, b^{|i}\ )$ specify the *architecture* of the ReLU NN. Therefore, we will denote the **architecture** of the ReLU NN $\mathscr{NN}_\Theta$ by $\text{Arch}(\Theta) \triangleq ((n, \mathfrak{o}_1), (\mathfrak{i}_2, \mathfrak{o}_2), \ldots, (\mathfrak{i}_K, m))$.

4.2.4 Special NN Operations

Definition 4.3 (Sequential (Functional) Composition)**.** *Let $\mathscr{NN}_{\Theta_1}$ and $\mathscr{NN}_{\Theta_2}$ be two NNs where $\texttt{Last}(Arch(\Theta_1)) = (\mathfrak{i}, \mathfrak{c})$ and $\texttt{First}(Arch(\Theta_2)) = (\mathfrak{c}, \mathfrak{o})$. Then the **functional composition** of $\mathscr{NN}_{\Theta_1}$ and $\mathscr{NN}_{\Theta_2}$, i.e. $\mathscr{NN}_{\Theta_1} \circ \mathscr{NN}_{\Theta_2}$, is a well defined NN, and can be represented by the parameter list $\Theta_1 \circ \Theta_2 \triangleq \texttt{Concat}(\Theta_1, \Theta_2)$.*

Definition 4.4. *Let $\mathscr{NN}_{\Theta_1}$ and $\mathscr{NN}_{\Theta_2}$ be two K-layer NNs with parameter lists: $\Theta_i = ((W_i^{|1}, b_i^{|1}), \ldots, (W_i^{|K}, b_i^{|K})), i = 1, 2$. Then the **parallel composition** of $\mathscr{NN}_{\Theta_1}$ and $\mathscr{NN}_{\Theta_2}$ is a NN given by the parameter list*

$$\Theta_1 \parallel \Theta_2 \triangleq \left(\left(\begin{bmatrix} W_1^{|1} \\ W_2^{|1} \end{bmatrix}, \begin{bmatrix} b_1^{|1} \\ b_2^{|1} \end{bmatrix} \right), \ldots, \left(\begin{bmatrix} W_1^{|K} \\ W_2^{|K} \end{bmatrix}, \begin{bmatrix} b_1^{|K} \\ b_2^{|K} \end{bmatrix} \right) \right). \tag{4.2}$$

That is $\Theta_1 \| \Theta_2$ accepts an input of the same size as (both) Θ_1 and Θ_2, but has as many outputs as Θ_1 and Θ_2 combined.

Definition 4.5 (n-element `min`/`max` NNs)**.** *An n**-element*** min ***network** is denoted by the parameter list $\Theta_{\min_n}$. $\mathscr{NN}(\Theta_{\min_n}) : \mathbb{R}^n \to \mathbb{R}$ such that $\mathscr{NN}(\Theta_{\min_n})(x)$ is the minimum from among the components of x (i.e. minimum according to the usual order relation $<$ on $\mathbb{R}$). An n**-element*** max ***network** is denoted by $\Theta_{\max_n}$, and functions analogously. These networks are described in [13].*

[1]That is Θ is not the concatenation of the $\theta^{(i)}$ into a single large matrix, so it preserves information about the sizes of the constituent $\theta^{(i)}$.

4.2.5 Two-Level-Lattice (TLL) Neural Networks

In this chapter, we will be especially concerned with ReLU NNs that have the Two-Level Lattice (TLL) architecture, as introduced with the AReN algorithm in [13]. Thus we define a TLL NN as follows.

Definition 4.6 (TLL NN [13, Theorem 2])**.** *A NN that maps $\mathbb{R}^n \to \mathbb{R}$ is said to be* ***TLL NN of size*** (N, M) *if the size of its parameter list $\Xi_{N,M}$ can be characterized entirely by integers N and M as follows.*

$$\Xi_{N,M} \triangleq \Theta_{\max_M} \circ \big((\Theta_{\min_N} \circ \Theta_{S_1}) \parallel \dots \parallel (\Theta_{\min_N} \circ \Theta_{S_M}) \big) \circ \Theta_\ell \tag{4.3}$$

where

- $\Theta_\ell \triangleq ((W_\ell, b_\ell))$*;*
- *each Θ_{S_j} has the form $\Theta_{S_j} = \big(S_j, \mathbf{0}_{M,1}\big)$; and*
- $S_j = \big[\llbracket I_N \rrbracket_{\iota_1,\cdot}^T \; \dots \; \llbracket I_N \rrbracket_{\iota_N,\cdot}^T \big]^T$ *for some sequence $\iota_k \in \{1, \dots, N\}$, where I_N is the $(N \times N)$ identity matrix.*

The matrices Θ_ℓ will be referred to as the ***linear function matrices*** *of $\Xi_{N,M}$. The matrices $\{S_j | j = 1, \dots, M\}$ will be referred to as the* ***selector matrices*** *of $\Xi_{N,M}$. Each set $s_j \triangleq \{k \in \{1, \dots, N\} | \exists \iota \in \{1, \dots, N\}. \llbracket S_j \rrbracket_{\iota,k} = 1\}$ is said to be the selector set of S_j.*

A ***multi-output TLL NN*** *with range space $\mathbb{R}^m$ is defined using m equally sized scalar TLL NNs. That is we denote such a network by $\Xi_{N,M}^{(m)}$, with each output component denoted by $\Xi_{N,M}^i$, $i = 1, \dots, m$.*

4.3 Problem Formulation

The main problem we consider in this chapter is one of TLL NN *repair*. In brief, we take as a starting point a TLL NN controller that is "mostly" correct in the sense that is provably safe under a specific set of circumstances (states); here we assume that safety entails avoiding a *particular, fixed subset of the state space.* However, we further suppose that this TLL NN controller induces some additional, *unsafe* behavior of (4.1) that is explicitly observed, such as from a more expansive application of a model checker; of course this unsafe behavior necessarily occurs in states not covered by the original safety guarantee. The repair problem, then, is to "repair" the given TLL controller so that this additional unsafe behavior is made safe, while simultaneously preserving the original safety guarantees associated with the network.

The basis for the problem in this chapter is thus a TLL NN controller that has been designed (or trained) to control (4.1) in a *safe way.* In particular, we use the following definition to fix our notion of "unsafe" behavior for (4.1).

Definition 4.7 (Unsafe Operation of (4.1))**.** *Let G_u be an $(\mathsf{K}_u \times n)$ real-valued matrix, and let h_u be an $(\mathsf{K}_u \times 1)$ real vector, which together define a set of* ***unsafe states*** $X_{\mathsf{unsafe}} \triangleq \{x \in \mathbb{R}^n | G_u x \geq h_u\}$.

Then, we mean that a TLL NN controller is safe with respect to (4.1) and X_{unsafe} in the following sense.

Definition 4.8 (Safe TLL NN Controller)**.** *Let $X_{\mathsf{safe}} \subset \mathbb{R}^n$ be a set of states such that $X_{\mathsf{safe}} \cap X_{\mathsf{unsafe}} = \emptyset$. Then a TLL NN controller $\mathfrak{u} \triangleq \mathscr{N\!N}(\Xi^{(m)}_{N,M}) : \mathbb{R}^n \to \mathbb{R}^m$ is* ***safe*** *for* (4.1) ***on horizon*** *T (with respect to X_{safe} and X_{unsafe}) if:*

$$\forall x_0 \in X_{\mathsf{safe}}, i \in \{1, ..., T\}. \big(\zeta_i^{x_0}(\mathscr{N\!N}(\Xi^{(m)}_{N,M})) \notin X_{\mathsf{unsafe}}\big). \tag{4.4}$$

That is $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$ *is safe (w.r.t.* $X_{\textsf{safe}}$*) if all of its length-T trajectories starting in* $X_{\textsf{safe}}$ *avoid the unsafe states* $X_{\textsf{unsafe}}$*.*

The design of safe controllers in the sense of Definition 4.8 has been considered in a number of contexts; see e.g. [43]. Often this design procedure involves training the NN using data collected from an expert, and verifying the result using one of many available NN verifiers [43].

However, as noted above, we further suppose that a given TLL NN which is safe in the sense of Definition 4.8 nevertheless has some *unsafe* behavior for states that lie outside $X_{\textsf{safe}}$. In particular, we suppose that a model checker (for example) provides to us a *counterexample* (or witness) to unsafe operation of (4.1).

Definition 4.9 (Counterexample to Safe Operation of (4.1))**.** *Let* $X_{\textsf{safe}} \subset \mathbb{R}^n$*, and let* $\mathfrak{u} \triangleq \mathscr{N\!N}(\Xi^{(m)}_{N,M})$ *be a TLL controller that is safe for* (4.1) *on horizon* T *w.r.t* $X_{\textsf{safe}}$ *and* $X_{\textsf{unsafe}}$*. A* ***counter example to the safe operation of*** (4.1) *is a state* $x_{\textsf{c.e.}} \notin X_{\textsf{safe}}$ *such that*

$$f(x_{\textsf{c.e.}}) + g(x_{\textsf{c.e.}}) \cdot \mathfrak{u}(x_{\textsf{c.e.}}) = \zeta_1^{x_{\textsf{c.e.}}}(\mathfrak{u}) \in X_{\textsf{unsafe}}. \tag{4.5}$$

That is starting (4.1) *in* $x_{\textsf{c.e.}}$ *results in an unsafe state in the next time step.*

We can now state the main problem of this chapter.

Problem 4.1. *Let dynamics* (4.1) *be given, and assume its trajectories are confined to compact subset of states,* $X_{\textsf{ws}}$ *(see Definition 4.2). Also, let* $X_{\textsf{unsafe}} \subset X_{\textsf{ws}}$ *be a specified set of unsafe states for* (4.1)*, as in Definition 4.7. Furthermore, let* $\mathfrak{u} = \mathscr{N\!N}(\Xi^{(m)}_{N,M})$ *be a TLL NN controller for* (4.1) *that is safe on horizon* T *with respect to a set of states* $X_{\textsf{safe}} \subset X_{\textsf{ws}}$ *(see Definition 4.8), and let* $x_{\textsf{c.e.}}$ *be a counterexample to safety in the sense of Definition 4.9.*

*Then the **TLL repair problem** is to obtain a new TLL controller $\bar{\mathfrak{u}} = \mathscr{N\!N}(\overline{\Xi}^{(m)}_{N,M})$ with the following properties:*

(i) $\overline{\mathfrak{u}}$ *is also safe on horizon T with respect to X_{safe};*

(ii) the trajectory $\zeta_1^{x_{\mathsf{c.e.}}}(\overline{\mathfrak{u}})$ is safe – i.e. the counterexample $x_{\mathsf{c.e.}}$ is "repaired";

(iii) $\overline{\Xi}^{(m)}_{N,M}$ and $\Xi^{(m)}_{N,M}$ share a common architecture (as implied by their identical architectural parameters); and

(iv) the selector matrices of $\overline{\Xi}^{(m)}_{N,M}$ and $\Xi^{(m)}_{N,M}$ are identical – i.e. $\overline{S}_k = S_k$ for $k = 1, \ldots, M$; and

(v) $\|\overline{W}_\ell - W_\ell\|_2 + \|\bar{b}_\ell - b_\ell\|_2$ is minimized.

In particular, *iii)*, *iv)* and *v)* justify the designation of this problem as one of "repair". That is the repair problem is to fix the counterexample while keeping the network as close as possible to the original network under consideration. **Note:** the formulation of Problem 4.1 only allows repair by means of altering the *linear layers* of $\Xi^{(m)}_{N,M}$; c.f. *(iii)* and *(iv)*.

4.4 Framework

The TLL NN repair problem described in Problem 4.1 is challenging because it has two main objectives, which are at odds with each other. In particular, repairing a counterexample requires altering the NN's response in a *local* region of the state space, but changing even a few neurons generally affects the *global* response of the NN – which could undo the initial safety guarantee supplied with the network. This tension is especially relevant for general deep NNs, and repairs realized on neurons in their latter layers. It is for this reason that we posed Problem 4.1 in terms of TLL NNs: our approach will be to use the unique semantics of TLL NNs to balance the trade-offs between **local NN alteration to repair the defective controller** and **global NN alteration to ensure that the repaired controller activates at the counterexample**. Moreover, locally repairing the defective controller at $x_{\mathsf{c.e.}}$ entails a further trade off between two competing objectives of its own: actually repairing the counterexample – Problem 4.1*(ii)* – without causing a violation of the original safety guarantee for X_{safe} – i.e. Problem 4.1*(i)*. Likewise, global alteration of the TLL to ensure correct activation of our repairs will entail its own trade-off: the alterations necessary to achieve the correct activation will also have to be made without sacrificing the safety guarantee for X_{safe} – i.e. Problem 4.1*(i)*.

We devote the remainder of this section to two crucial subsections, one for each side of this local/global dichotomy. Our goal in these two subsections is to describe **constraints** on a TLL controller that are **sufficient** to ensure that it accomplishes the repair described in Problem 4.1. Thus, the results in this section should be seen as optimization constraints around which we can build our algorithm to solve Problem 4.1. The algorithmic details and formalism are presented in Section 4.5.

4.4.1 Local TLL Repair

We first consider in isolation the problem of repairing the TLL controller in the vicinity of the counterexample $x_{\mathsf{c.e.}}$, but under the assumption that the altered controller will remain the active there. The problem of actually guaranteeing that this is the case will be considered in the subsequent section. Thus, we proceed with the repair by **establishing constraints** on the alterations of those parameters in the TLL controller associated with the affine controller instantiated at and around the state $x_{\mathsf{c.e.}}$. To be consistent with the literature, we will refer to any individual affine function instantiated by a NN as one of its *local linear functions.*

Definition 4.10 (Local Linear Function)**.** *Let* $\mathsf{f} : \mathbb{R}^n \to \mathbb{R}$ *be CPWA. Then a* ***local linear function of*** f *is a linear function* $\ell : \mathbb{R}^n \to \mathbb{R}$ *if there exists an open set* $\mathfrak{O}$ *such that* $\ell(x) = \mathsf{f}(x)$ *for all* $x \in \mathfrak{O}$*.*

The unique semantics of TLL NNs makes them especially well suited to this local repair task because in a TLL NN, its local linear functions appear directly as neuronal parameters. In particular, all of the local linear functions of a TLL NN are described *directly* by parameters in its linear layer; i.e. $\Theta_\ell = (W_\ell, b_\ell)$ for scalar TLL NNs or $\Theta^\kappa_\ell = (W^\kappa_\ell, b^\kappa_\ell)$ for the κ^{th} output of a multi-output TLL (see Definition 4.6). This follows as a corollary of the following relatively straightforward proposition, borrowed from [15]:

Proposition 4.1 ([15, Proposition 3])**.** *Let* $\Xi_{N,M}$ *be a scalar TLL NN with linear function matrices* $\Theta_\ell = (W_\ell, b_\ell)$*. Then every local linear function of* $\mathscr{N\!N}(\Xi_{N,M})$ *is exactly equal to* $\ell_i : x \mapsto [\![W_\ell x + b_\ell]\!]_{i,\cdot}$ *for some* $i \in \{1, \ldots, N\}$*.*

Similarly, let $\Xi^{(m)}_{N,M}$ *be a multi-output TLL, and let* ℓ *be any local linear function of* $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$*. Then for each* $\kappa \in \{1, \ldots, m\}$*, the* κ^{th} *component of* ℓ *satisfies* $[\![\ell]\!]_{\kappa,\cdot} = x \mapsto [\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{i_\kappa,\cdot}$ *for some* $i_\kappa \in \{1, \ldots, N\}$*.*

Corollary 4.1. *Let* $\Xi^{(m)}_{N,M}$ *be a TLL over domain* $\mathbb{R}^n$*, and let* $x_{\mathsf{c.e.}} \in \mathbb{R}^n$*. Then there exist* m *integers* $act_k \in \{1, \ldots, N\}$ *for* $k = 1, \ldots, m$ *and a closed, connected set with non-empty interior,* $R_a \subset \mathbb{R}^n$ *such that*

- $x_{\mathsf{c.e.}} \in R_a$*; and*
- $[\![\mathscr{N\!N}(\Xi^{(m)}_{N,M})]\!]_k = x \mapsto [\![W^k_\ell x + b^k]\!]_{\text{act}_k}$ *on the set* R_a*.*

Proof. It is straightforward to see that every point x in the domain of $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$ belongs to the closure of some open set $\mathfrak{O}$, on which $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$ is affine (i.e. equal to one of its local linear functions). For if this weren't the case, then there would be an open subset of the domain of $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$, where it *wasn't* affine, thus contradicting the CPWA property of a ReLU NN.

Thus, let $\mathfrak{D}_{x_{\text{c.e.}}}$ be such an open set that includes $x_{\text{c.e.}}$ in its closure, and let $\ell : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be the local linear function of $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$ on $\mathfrak{D}_{x_{\text{c.e.}}}$. We can further assume that $\mathfrak{D}_{x_{\text{c.e.}}}$ is connected without loss of generality, so set $R_a = \overline{\mathfrak{D}}_{x_{\text{c.e.}}}$.

By Proposition 4.1, there exists indices $\{\text{act}_\kappa\}_{\kappa=1}^m$ such that

$$[\![\ell]\!]_{\text{act}_\kappa} = x \mapsto [\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{\text{act}_\kappa}. \tag{4.6}$$

But by the definition of ℓ and the above, we also have that

$$\forall x \in \mathfrak{D}_{\text{c.e.}} \,.\, [\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{\text{act}_\kappa} = [\![\mathscr{N\!N}(\Xi^{(m)}_{N,M})(x)]\!]_{\text{act}_\kappa}. \tag{4.7}$$

Thus, the conclusion of the corollary holds on $\mathfrak{D}_{x_{\text{c.e.}}}$; it holds on $R_a = \overline{D}_{x_{\text{c.e.}}}$ by continuity of $\mathscr{N\!N}(\Xi^{(m)}_{N,M})$. □

Corollary 4.1 is actually a strong statement: it indicates that in a TLL, each local linear function is described directly by *its own* linear-function-layer parameters and those parameters describe *only* that local linear function.

Thus, as a consequence of Corollary 4.1, "repairing" the problematic local controller (local linear function) of the TLL controller in Problem 4.1 involves the following steps:

1. identify which of the local linear functions is realized by the TLL controller at $x_{\text{c.e.}}$ – i.e. identifying the indices of the active local linear function at $x_{\text{c.e.}}$ viz. indices $\text{act}_\kappa \in \{1, \dots, N\}$ for each output κ as in Corollary 4.1;

2. *establish constraints* on the parameters of that local linear function so as to ensure re-

pair of the counterexample; i.e. altering the elements of the rows $[\![W_\ell^\kappa]\!]_{\text{act}_\kappa,\cdot}$ and $[\![b_\ell^\kappa]\!]_{\text{act}_\kappa}$ for each output κ such that the resulting linear controller repairs the counterexample as in Problem 4.1*(ii)*; and

3. *establish constraints* to ensure the repaired parameters do not induce a violation of the safety constraint for the guaranteed set of safe states, X_{safe}, as in Problem 4.1*(i)*.

We consider these three steps in sequence as follows.

1) Identifying the Active Controller at $x_{\mathsf{c.e.}}$: From Corollary 4.1, *all* of the possible linear controllers that a TLL controller realizes are exposed directly in the parameters of its linear layer matrices, Θ_ℓ^κ. Crucially for the repair problem, once the active controller at $x_{\mathsf{c.e.}}$ has been identified, the TLL parameters responsible for that controller immediately evident. This is the starting point for our repair process.

Since a TLL consists of two levels of lattice operations, it is straightforward to identify which of these affine functions is in fact active at $x_{\mathsf{c.e.}}$; for a given output, κ, this is can be done by evaluating $W_\ell^\kappa x_{\mathsf{c.e.}} + b_\ell^\kappa$ and comparing the components thereof according to the selector sets associated with the TLL controller. That is the index of the active controller for output κ, denoted by act_κ, is determined by the following two expressions:

$$\mu_k^\kappa \triangleq \arg\min_{i \in S_k^\kappa} [\![W_\ell^\kappa x_{\mathsf{c.e.}} + b_\ell^\kappa]\!]_i \tag{4.8}$$

$$\text{act}_\kappa \triangleq \arg\max_{j \in \{\mu_k^\kappa | k=1,\dots,M\}} [\![W_\ell^\kappa x_{\mathsf{c.e.}} + b_\ell^\kappa]\!]_j \tag{4.9}$$

These expressions mirror the computations that define a TLL network, as described in Definition 4.6; the only difference is that `max` and `min` are replaced by `arg max` and `arg min`, respectively, so as to retrieve the index of interest instead of the network's output.

2) Repairing the Affine Controller at $x_{\mathsf{c.e.}}$: Given the result of Corollary 4.1, the parameters of the network that result in a problematic controller at $x_{\mathsf{c.e.}}$ are readily apparent. Moreover, since these parameters are obviously in the linear layer of the original TLL, they are alterable under the requirement in Problem 4.1 that only linear-layer parameters are permitted to be used for repair. Thus, in the current context, local repair entails simply correcting the elements of the matrices $[\![W_\ell^k]\!]_{\text{act}_k}$ and $[\![b_\ell^k]\!]_{\text{act}_k}$. It is thus clear that a "repaired" controller should satisfy

$$f(x_{\mathsf{c.e.}}) + g(x_{\mathsf{c.e.}}) \left[\begin{smallmatrix} [\![W_\ell^1 x_{\mathsf{c.e.}} + b_\ell^1]\!]_{\text{act}_1} \\ \vdots \\ [\![W_\ell^m x_{\mathsf{c.e.}} + b_\ell^m]\!]_{\text{act}_m} \end{smallmatrix} \right] \not\in X_{\mathsf{unsafe}}. \tag{4.10}$$

Then (4.10) represents a *linear constraint* in the local controller to be repaired, and this constraint imposes the repair property in Problem 4.1*(ii)*. That is provided that the repaired controller described by $\{\text{act}_\kappa\}$ *remains active* at the counterexample; as noted, we consider this problem in the global stasis condition subsequently.

3) Preserving the Initial Safety Condition with the Repaired Controller: One unique aspect of the TLL NN architecture is that affine functions defined in its linear layer can be *reused* across regions of its input space. In particular, the controller associated with the parameters we repaired in the previous step – i.e. the indices $\{\text{act}_\kappa\}$ of the linear layer matrices – may likewise be activated in or around X_{safe}. The fact that we *altered* these controller parameters thus means that trajectories emanating from X_{safe} may be affected in turn by our repair efforts: that is the repairs we made to the controller to address Problem 4.1*(ii)* may simultaneously alter the TLL in a way that **undoes** the requirement in Problem 4.1*(i)* – i.e. the initial safety guarantee on X_{safe} and $\mathscr{NN}(\Xi_{N,M}^{(m)})$. Thus, local repair of the problematic controller must account for this safety property, too.

We accomplish this by bounding the reach set of (4.1) for initial conditions in X_{safe}, and for this we employ the usual strategy of bounding the relevant Lipschitz constants.

Naturally, since the TLL controller is a CPWA controller operated in closed loop, these bounds will also incorporate the size of the TLL controller parameters $\|[\![W_\ell^\kappa]\!]_i\|$ and $\|[\![b_\ell^\kappa]\!]_i\|$ for $\kappa \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, N\}$.

In general, however, we have the following proposition.

Proposition 4.2. *Consider system dynamics* (4.1)*, and suppose that the state x is confined to known compact workspace, X_{ws} (see Definition 4.2). Also, let T be the integer time horizon from Definition 4.8. Finally, assume that a closed-loop CPWA $\Psi : \mathbb{R}^n \to \mathbb{R}^m$ is applied to* (4.1)*, and that Ψ has local linear functions $\mathcal{L}_\Psi = \{x \mapsto w_k x + b_k | k = 1, \ldots, N\}$.*

Moreover, define the function β as

$$\beta(\|w\|, \|b\|) \triangleq \sup_{x_0 \in X_{safe}} \Big(\|f(x_0) - x_0\| + \\ \|g(x_0)\| \cdot \|w\| \cdot ext(X_{ws}) + \|g(x_0)\| \cdot \|b\| \Big) \tag{4.11}$$

and in turn define

$$\beta_{max}(\Psi) \triangleq \beta\big(\max_{w \in \{w_k | k=1,\ldots,N\}} \|w\|, \max_{b \in \{b_k | k=1,\ldots,N\}} \|b\| \big). \tag{4.12}$$

Finally, define the function L as in (4.11)*, and in turn define*

$$L_{max}(\Psi) \triangleq L\big(\max_{w \in \{w_k | k=1,\ldots,N\}} \|w\|, \max_{b \in \{b_k | k=1,\ldots,N\}} \|b\| \big). \tag{4.2}$$

$$L(\|w\|, \|b\|) \triangleq L_f + L_g \cdot \sup_{x_0 \in X_{\mathsf{safe}}} \|w\| \cdot \|x_0\| + \sup_{x_0 \in X_{\mathsf{safe}}} \|w\| \cdot \|g(x_0)\| + L_g \cdot \|b\| \tag{4.11}$$

Then for all $x_0 \in X_{\mathsf{safe}}$, $i \in \{1, \ldots, T\}$, we have:

$$\|\zeta_T^{x_0}(\Psi) - x_0\| \leq \beta_{max}(\Psi) \cdot \sum_{k=0}^{T} L_{max}(\Psi)^k. \tag{4.3}$$

Lemma 4.1. *Let $F : x \mapsto g(x) \cdot u(x)$ for Lipschitz continuous functions $g : \mathbb{R}^n \to \mathbb{R}^{(n \cdot m)}$ (with output an $(n \times m)$ real-valued matrix) and $u : \mathbb{R}^n \to \mathbb{R}^m$ with Lipschitz constants L_g and L_u, respectively.*

Then on compact subset $X \subset \mathbb{R}^n$, F is Lipschitz continuous with Lipschitz constant $L_F = L_g \cdot \sup_{x \in X}\|u(x)\| + L_u \cdot \sup_{x \in X}\|g(x)\|$.

Proof. This follows by straightforward manipulations as follows.

Let $x, x' \in X$ and note that:

$$\begin{aligned}
&\|g(x)u(x) - g(x')u(x')\| \\
&= \|g(x)u(x) + (-g(x') + g(x'))u(x) - g(x')u(x')\| \\
&= \|(g(x) - g(x'))u(x) + g(x')(u(x) - u(x'))\| \\
&\leq \|g(x) - g(x')\| \cdot \|u(x)\| + \|u(x) - u(x')\| \cdot \|g(x')\| \\
&\leq \big(L_g \cdot \|u(x)\| + L_u \cdot \|g(x')\|\big) \cdot \|x - x'\| \\
&\leq \big(L_g \cdot \sup_{x \in X}\|u(x)\| + L_u \cdot \sup_{x' \in X}\|g(x')\|\big) \cdot \|x - x'\|.
\end{aligned}$$

Proof. (Proposition 4.2) We will expand and bound the quantity on the left-hand side of the conclusion, (4.3).

$$\begin{aligned}
&\|\zeta_T^{x_0}(\Psi) - x_0\| \\
&\quad = \|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi) + \zeta_{T-1}^{x_0}(\Psi) - x_0\| \\
&\quad \leq \|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi)\| + \|\zeta_{T-1}^{x_0}(\Psi) - x_0\| \qquad (4.4)
\end{aligned}$$

We then bound the first term as follows:

$$\begin{aligned}
&\|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi)\| \\
&\quad \leq \|f(\zeta_{T-1}^{x_0}(\Psi)) - f(\zeta_{T-2}^{x_0}(\Psi))\| \\
&+ \Big\| g(\zeta_{T-1}^{x_0}(\Psi)) \cdot \big[w(\zeta_{T-1}^{x_0}(\Psi)) \cdot \zeta_{T-1}^{x_0}(\Psi) + b(\zeta_{T-1}^{x_0}(\Psi)) \big] \\
&- g(\zeta_{T-2}^{x_0}(\Psi)) \cdot \big[w(\zeta_{T-2}^{x_0}(\Psi)) \cdot \zeta_{T-2}^{x_0}(\Psi) + b(\zeta_{T-2}^{x_0}(\Psi)) \big] \Big\|
\end{aligned} \tag{4.5}$$

where the functions $w : \mathbb{R}^n \to \mathbb{R}^n$ and $b : \mathbb{R}^n \to \mathbb{R}$ return a (unique) choice of the linear (weights) and affine (bias) of the local linear function of Ψ that is active at their argument.

Now, we collect the $w(\cdot)$ and $b(\cdot)$ terms in right-hand side of (4.5). That is:

$$\begin{aligned}
&\|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi)\| \\
&\quad \leq \|f(\zeta_{T-1}^{x_0}(\Psi)) - f(\zeta_{T-2}^{x_0}(\Psi))\| \\
&+ \Big\| g(\zeta_{T-1}^{x_0}(\Psi)) w(\zeta_{T-1}^{x_0}(\Psi)) \zeta_{T-1}^{x_0}(\Psi) \\
&\qquad\qquad - g(\zeta_{T-2}^{x_0}(\Psi) w(\zeta_{T-2}^{x_0}(\Psi)) \zeta_{T-2}^{x_0}(\Psi) \Big\| \\
&+ \Big\| g(\zeta_{T-1}^{x_0}(\Psi)) b(\zeta_{T-1}^{x_0}(\Psi)) - g(\zeta_{T-2}^{x_0}(\Psi) b(\zeta_{T-2}^{x_0}(\Psi)) \Big\|
\end{aligned}$$

The first term in the above can be directly bounded using the Lipschitz constant of f. Also, since there are only finitely many local linear function of Ψ, $b(\cdot)$ takes one of finitely many values across the entire state space, and we may bound the associated term using this observation. Finally, we can Lemma 4.1 to the second term, noting that the linear function defined by $w(\cdot)$ has Lipschitz constant $\|w(\cdot)\|$ and there are only finitely many possible values for this quantity (one for each local linear function). This yields the following bound:

$$\begin{aligned}
&\|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi)\| \\
&\quad \leq L_f \cdot \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\| \\
&+ \big(L_g \cdot \sup_{x \in X_{\mathsf{ws}}} \|w(x) \cdot x\| + \max_k \|w_k\| \sup_{x \in X_{\mathsf{ws}}} \|g(x)\|\big) \\
&\qquad\qquad \cdot \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\| \\
&+ \max_k \|b_k\| \cdot L_g \cdot \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\|
\end{aligned}$$

If we simplify, then we see that we have

$$\|\zeta_T^{x_0}(\Psi) - \zeta_{T-1}^{x_0}(\Psi)\| \leq L_{\max}(\Psi) \cdot \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\| \tag{4.6}$$

with $L_{\max}(\Psi)$ as defined in the statement of the Proposition.

Now, we expand the final term of (4.4) as

$$\|\zeta_{T-1}^{x_0}(\Psi) - x_0\| \leq \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\| + \|\zeta_{T-2}^{x_0}(\Psi) - x_0\| \tag{4.7}$$

so that (4.4) can be rewritten as:

$$\|\zeta_T^{x_0}(\Psi) - x_0\| \leq (L_{\max}(\Psi) + 1) \cdot \|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\| + \|\zeta_{T-2}^{x_0}(\Psi) - x_0\|. \tag{4.8}$$

But now we can proceed inductively, applying the bound (4.6) mutatis mutandis to the expression $\|\zeta_{T-1}^{x_0}(\Psi) - \zeta_{T-2}^{x_0}(\Psi)\|$ in (4.8). This induction can proceed until the factor to be expanded using (4.6) has the form $\|\zeta_{T-(T-1)}^{x_0}(\Psi) - \zeta_{T-(T)}^{x_0}(\Psi)\|$, which will yield the bound:

$$\|\zeta_T^{x_0}(\Psi) - x_0\| \leq \|\zeta_1^{x_0}(\Psi) - x_0\| \cdot \sum_{k=0}^{T} L_{\max}(\Psi)^k. \tag{4.9}$$

Thus it remains to bound the quantity $\|\zeta_1^{x_0}(\Psi) - x_0\|$. We proceed to do this in a relatively straightforward way:

$$\begin{aligned}
&\|\zeta_1^{x_0}(\Psi) - x_0\| \\
&= \|f(x_0) + g(x_0)\,[w(x_0)x_0 + b(x_0)] - x_0\| \\
&\leq \|f(x_0) - x_0\| + \|g(x_0)\| \cdot \|w(x_0)\| \cdot \|x_0\| \\
&\qquad\quad + \|g(x_0)\| \cdot \|b(x_0)\|.
\end{aligned} \tag{4.10}$$

Finally, since we're interested in bounding the original quantity, $\|\zeta_T^{x_0}(\Psi) - x_0\|$, over all $x_0 \in X_{\mathsf{safe}}$, we can upper-bound the above by taking a supremum over all $x_0 \in X_{\mathsf{safe}}$. Thus,

$$\sup_{x \in X_{\mathsf{safe}}} \|\zeta_T^{x_0}(\Psi) - x_0\| \leq \sup_{x \in X_{\mathsf{safe}}} \|\zeta_1^{x_0}(\Psi) - x_0\| \cdot \sum_{k=0}^{T} L_{\max}(\Psi)^k \tag{4.11}$$

where the sup on the right-hand side does not interact with the summation, since $L_{\max}(\Psi)$ is constant with respect to x_0. The final conclusion is obtained by observing that

$$\sup_{x \in X_{\mathsf{safe}}} \|\zeta_1^{x_0}(\Psi) - x_0\| \leq \beta_{\max}(\Psi) \tag{4.12}$$

with $\beta_{\max}(\Psi)$ as defined in the statement of the proposition. □

Proposition 4.2 bounds the size of the reach set for (4.1) in terms of an arbitrary CPWA controller, Ψ, when the system is started from X_{safe}. This proposition is naturally applied in order to find bounds for safety with respect to the *unsafe* region X_{unsafe} as follows.

Proposition 4.3. *Let T, X_{ws}, Ψ and $\mathcal{L}_\Psi$ be as in Proposition 4.2, and let β_{max} and L_{max} be two constants s.t. for all $\delta x \in \mathbb{R}^n$*

$$\|\delta x\| \leq \beta_{max} \cdot \sum_{k=0}^{T} {L_{max}}^k \implies \forall x_0 \in X_{\mathsf{safe}}.\big(x_0 + \delta x \notin X_{\mathsf{unsafe}}\big) \tag{4.13}$$

If $\beta_{max}(\Psi) \leq \beta_{max}$ and $L_{max}(\Psi) \leq L_{max}$, then trajectories of (4.1) *under closed loop controller Ψ are safe in the sense that*

$$\forall x_0 \in X_{\mathsf{safe}} \forall i \in \{1, \dots, T\} \, . \, \zeta_t^{x_0}(\Psi) \notin X_{\mathsf{unsafe}}. \tag{4.14}$$

Proof. This is more or less a straightforward application of Proposition 4.2.

Indeed, by Proposition 4.2 and the assumption of this proposition, we conclude that

$$\begin{aligned} \|\zeta_T^{x_0}(\Psi) - x_0\| &\leq \beta_{\max}(\Psi) \cdot \sum_{k=0}^{T} L_{\max}(\Psi)^k \\ &\leq \beta_{\max} \cdot \sum_{k=0}^{T} {L_{\max}}^k. \end{aligned}$$

Hence, $\delta x = \zeta_T^{x_0}(\Psi) - x_0$ triggers the implication in (4.13), and we conclude that

$$\forall x_0 \in X_{\mathsf{safe}} \, . \, x_0 + \delta x = \zeta_T^{x_0}(\Psi) \notin X_{\mathsf{unsafe}} \tag{4.15}$$

as required. $\square$

In particular, Proposition 4.3 states that if we find constants $\beta_{\max}$ and $L_{\max}$ that satisfy (4.13), then we have a way to bound the parameters of any CPWA controller (via β and L) so that that controller is safe in closed loop. This translates to conditions that our repaired controller must satisfy in order to preserve the safety property required in Problem 4.1*(i)*. Formally, this entails particularizing Proposition 4.2 and 4.3 to the TLL controllers associated with the repair problem.

Corollary 4.2. *Again consider system* (4.1) *confined to workspace X_{ws} as before. Also, let β_{max} and L_{max} be such that they satisfy the assumptions of Proposition 4.3, viz.* (4.13).

Now, let $\Xi_{N,M}^{(m)}$ be the TLL controller as given in Problem 4.1, and let $\Theta_\ell^\kappa = (W_\ell^\kappa, b_\ell^\kappa)$ be *its linear layer matrices for outputs $\kappa = 1, \ldots, m$ as usual. For this controller, define the following two quantities:*

$$\Omega_W \triangleq \max_{w \in \cup_{\kappa=1}^{m} \{ [\![W_\ell^\kappa]\!]_j | j=1,\ldots,N \}} \|w\| \tag{4.16}$$

$$\Omega_b \triangleq \max_{b \in \cup_{\kappa=1}^{m} \{ [\![b_\ell^\kappa]\!]_j | j=1,\ldots,N \}} \|b\| \tag{4.17}$$

so that $\beta_{max}(\Xi_{N,M}^{(m)}) = \beta(\Omega_W, \Omega_b)$ and $L_{max}(\Xi_{N,M}^{(m)}) = L(\Omega_W, \Omega_b)$. Finally, let indices $\{act_\kappa\}_{\kappa=1}^{m}$ specify the active local linear functions of $\Xi_{N,M}^{(m)}$ that are to be repaired, as described in Subsection 4.4.1.1 and 4.4.1.2. Let $\bar{w}_{act_\kappa}^{\kappa}$ and $\bar{b}_{act_\kappa}^{\kappa}$ be any repaired values of $[\![W_\ell^\kappa]\!]_{act_\kappa, \cdot}$ and $[\![b_\ell^\kappa]\!]_{act_\kappa}$, respectively.

If the following four conditions are satisfied

$$\beta(\|\bar{w}_{act_\kappa}^{\kappa}\|, \|\bar{b}_{act_\kappa}^{\kappa}\|) \leq \beta_{max} \tag{4.18}$$

$$\beta_{max}(\Xi_{N,M}^{(m)}) \leq \beta_{max} \tag{4.19}$$

$$L(\|\bar{w}_{act_\kappa}^{\kappa}\|, \|\bar{b}_{act_\kappa}^{\kappa}\|) \leq L_{max} \tag{4.20}$$

$$L_{max}(\Xi_{N,M}^{(m)}) \leq L_{max} \tag{4.21}$$

then the following hold for all $x_0 \in X_{\mathsf{safe}}$:

$$\|\zeta_T^{x_0}(\bar{\Xi}_{N,M}^{(m)}) - x_0\| \leq \beta_{max} \cdot \sum_{k=0}^{T} {L_{max}}^k \tag{4.22}$$

and hence

$$\forall i \in \{1, \ldots, T\} \,.\, \zeta_i^{x_0}(\bar{\Xi}_{N,M}^{(m)}) \notin X_{\mathsf{unsafe}}. \tag{4.23}$$

Proof. Corollary 4.2 is simply a particularization of Proposition 4.3 to the repaired TLL network, $\bar{\Xi}^{(m)}_{N,M}$. It is only necessary to note that we have separate conditions to ensure that conclusion of Proposition 4.3 applied both to the *original* TLL network (i.e. (4.19) and (4.21)), as well as the repaired TLL parameters (i.e. (4.18) and (4.20)). $\square$

The conclusion (4.22) of Corollary 4.2 should be interpreted as follows: the bound on the reach set of the repaired controller, $\bar{\Xi}^{(m)}_{N,M}$, is no worse than the bound on the reach set of the original TLL controller given in Problem 4.1. Hence, by the assumptions borrowed from Proposition 4.3, conclusion (4.23) of Corollary 4.2 indicates that **the repaired controller** $\bar{\Xi}^{(m)}_{N,M}$ **remains safe in the sense of** Problem 4.1*(i)* – i.e. closed-loop trajectories emanating from X_{safe} remain safe on horizon T. For the subsequent development of our algorithm, (4.18) and (4.20) will play the crucial role of ensuring that the repaired controller respects the guarantee of Problem 4.1*(i)*.

4.4.2 Global TLL Alteration for Repaired Controller Activation

In the context of local repair, we identified the local linear function instantiated by the TLL controller, and repaired the parameters associated with that particular function – i.e. the repairs were affected on a particular, indexed row of W^{κ}_{ℓ} and b^{κ}_{ℓ}. We then proceeded under the assumption that the affine function *at that index* would remain active in the output of the TLL network at the counterexample, *even* *<u>after</u>* altering its parameters. Unfortunately, this is not case in a TLL network per se, since the value of each local linear function at a point interacts with the selector matrices (see Definition 4.6) to determine whether it is active or not. In other words, changing the parameters of a particular indexed local linear function in a TLL will change its output value at any given point (in general), and hence also the region on which said indexed local linear function is *active*.

Analogous to the local alteration consider before, we thus need to **devise <u>global</u> constraints sufficient to enforce the activation of the repaired controller at $x_{\text{c.e.}}$.**

This observation is manifest in the computation structure that defines a TLL NN: a particular affine function is active in the output of the TLL if and only if it is active in the output of one of the `min` networks (see Definition 4.6), and the output of that same `min` network exceeds the output of all others, thereby being active at the output of the final `max` network (again, see Definition 4.6). Thus, ensuring that a particular, indexed local linear function is active at the output of a TLL entails ensuring that that function

(a) appears at the output of one of the `min` networks; and

(b) appears at the output of the `max` network, by exceeding the outputs of all the *other* `min` networks.

Notably, this sequence also suggests a mechanism for meeting the task at hand: ensuring that the repaired controller remains active at the counter example.

Formally, we have the following proposition.

Proposition 4.4. *Let $\Xi^{(m)}_{N,M}$ be a TLL NN over $\mathbb{R}^n$ with output-component linear function matrices $\Theta^{\kappa}_{\ell} = (W^{\kappa}_{\ell}, b^{\kappa}_{\ell})$ as usual, and let $x_{\text{c.e.}} \in \mathbb{R}^n$.*

Then the index $act_\kappa \in \{1, \dots, N\}$ denote the local linear function that is active at $x_{\text{c.e.}}$ for output κ, as described in Corollary 4.1, if and only if there exists index $sel_\kappa \in \{1, \dots, M\}$ such that

(i) for all $i \in S^{\kappa}_{sel_\kappa}$ and any $x \in R_a$,

$$[\![W^{\kappa}_{\ell} x + b^{\kappa}_{\ell}]\!]_{act_\kappa,\cdot} \leq [\![W^{\kappa}_{\ell} x + b^{\kappa}_{\ell}]\!]_{i,\cdot} \tag{4.24}$$

i.e. the active local linear function "survives" the min *network associated with selector set* $S^\kappa_{sel_\kappa}$*; and*

(ii) for all $j \in \{1, \dots, M\}\backslash\{sel_\kappa\}$ *there exists an index* $\iota^\kappa_j \in \{1, \dots, N\}$ *s.t. for all* $x \in R_a$

$$[\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{\iota^\kappa_j,\cdot} \leq [\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{act_\kappa,\cdot} \tag{4.25}$$

i.e. the active local linear function "survives" the max *network of output* κ *by exceeding the output of all of the other* min *networks.*

This proposition follows calculations similar to those mentioned before.

Proof. The "if" portion of this proof is suggested by the computations in Section 4.4.4.1, so we focus on the "only if" portion.

Thus, let $\{\text{act}_\kappa\}^m_{\kappa=1} \in \{1, \dots, N\}^m$ be a set of indices, and assume that there exists an index $\{\text{sel}_\kappa\}^m_{\kappa=1} \in \{1, \dots, M\}^m$ for which the "only if" assumptions of the proposition are satisfied. We will show that the local linear function with indices $\{\text{act}_\kappa\}^m_{\kappa=1}$ is in fact active on R_a.

This will follow more or less directly by simply carrying out the computations of the TLL NN on R_a. In particular, by condition *(i)*, we have that $\mathscr{NN}(\!|\Theta_{\min_N} \circ \Theta_{S^\kappa_{\text{sel}_\kappa}} \circ \Theta^\kappa_\ell|\!) = x \mapsto [\![W^\kappa_\ell x + b^\kappa_\ell]\!]_{\text{act}_\kappa}$ for all $x \in R_a$, $\kappa = 1, \dots, m$. Then, by condition *(ii)* we have that for all $j \in \{1, \dots, M\}\backslash\{\text{sel}_\kappa\}$ and $x \in R_a$

$$\mathscr{NN}(\!|\Theta_{\min_N} \circ \Theta_{S^\kappa_{\text{sel}_\kappa}} \circ \Theta^\kappa_\ell|\!)(x) \leq \mathscr{NN}(\!|\Theta_{\min_N} \circ \Theta_{S^\kappa_{\text{sel}_\kappa}} \circ \Theta^\kappa_\ell|\!)(x). \tag{4.26}$$

The conclusion thus follows immediately from (4.26) and the fact that the `min` groups for the input to the final layer of the output's TLL, $\Theta_{\max_M}$. $\square$

The "only if" portion of Proposition 4.4 thus directly suggests constraints to impose such that the desired local linear function act_κ is active on its respective output. In particular, among the **non-active local linear functions at** $x_{\textsf{c.e.}}$, at least one must be altered from each of the selector sets $s_j : j \in \{1, \dots, M\}\backslash\{\text{sel}_\kappa\}$. The fact that these alterations must be made to local linear functions which are not active at the counterexample warrants the description of this procedure as "global alteration".

Finally, however, we note that altering these un-repaired local linear functions – i.e. those not indexed by act_κ – may create the same issue described in Section 4.3. Thus, for any of these global alterations additional safety constraints like (4.18) and (4.20) must be imposed on the altered parameters.

4.5 Main Algorithm

Problem 4.1 permits the alteration of linear-layer parameters in the original TLL controller to perform repair. In Section 4.4, we developed *constraints* on these parameters to perform

- first, local alteration to ensure repair of the defective controller at $x_{\textsf{c.e.}}$; and
- subsequently, global alteration to ensure that the repaired local controller is activated at and around $x_{\textsf{c.e.}}$.

The derivations of both sets of constraints implies that they are merely sufficient conditions for their respective purposes, so there is no guarantee that any subset of them are jointly feasible. Moreover, as a "repair" problem, any repairs conducted must involve minimal alteration – Problem 4.1*(v)*.

Thus, the core of our algorithm is to employ a convex solver to find the minimally altered TLL parameters that also satisfy the local and global constraints we have outlined for successful repair with respect to the other aspects of Problem 4.1. The fact that the local repair constraints are prerequisite to the global activation constraints means that we will employ a convex solver on two optimization problems *in sequence*: first, to determine the feasibility of local repair and effectuate that repair in a minimal way; and then subsequently to determine the feasibility of activating said repaired controller as required and effectuating that activation in a minimal way.

4.5.1 Optimization Problem for Local Alteration (Repair)

Local alteration for repair starts by identifying the active controller at the counterexample, as denoted by the index act_κ for each output of the controller, κ. The local controller for each output is thus the starting point for repair in our algorithm, as described in the prequel. From this knowledge, an explicit constraint sufficient to repair the local controller at $x_{\mathsf{c.e.}}$ is specified directly by the dynamics: see (4.10).

Our formulation of a safety constraint for the locally repaired controller requires additional input, though. In particular, we need to identify constants $\beta_{\max}$ and $L_{\max}$ such that the non-local controllers satisfy (4.19) and (4.21). Then Corollary 4.2 implies that (4.18) and (4.20) are constraints that ensure the repaired controller satisfies Problem 4.1*(i)*. For this we take the naive approach of setting $\beta_{\max} = \beta(\Xi^{(m)}_{N,M})$, and then solving for the smallest $L_{\max}$ that ensures safety for that particular $\beta_{\max}$. In particular, we set

$$L_{\max} = \inf\{L' > 0 \mid \beta_{\max} \cdot \sum_{k=0}^{T} {L'}^{k} = \inf_{\substack{x_s \in X_{\mathsf{safe}} \\ x_u \in X_{\mathsf{unsafe}}}} \|x_s - x_u\|\}. \tag{4.27}$$

Given this information the local repair optimization problem can be formulated for a multi-output TLL as:

$$\begin{aligned}
\texttt{Local}: \min_{\bar{w}^{\kappa}_{\text{act}_\kappa}, \bar{b}^{\kappa}_{\text{act}_\kappa}} \quad & \sum_{\kappa=1}^{m} \| [\![W^{\kappa}_{\ell}]\!]_{\text{act}_\kappa} - \bar{w}^{\kappa}_{\text{act}_\kappa} \| + \| [\![b^{\kappa}_{\ell}]\!]_{\text{act}_\kappa} - \bar{b}^{\kappa}_{\text{act}_\kappa} \| \\
\text{s.t.} \quad & f(x_{\mathsf{c.e.}}) + g(x_{\mathsf{c.e.}}) \begin{bmatrix} \bar{w}^{1}_{\text{act}_1} x_{\mathsf{c.e.}} + \bar{b}^{1}_{\text{act}_1} \\ \vdots \\ \bar{w}^{m}_{\text{act}_m} x_{\mathsf{c.e.}} + \bar{b}^{m}_{\text{act}_m} \end{bmatrix} \notin X_{\mathsf{unsafe}} \\
& \forall \kappa = 1, \ldots, m \;.\; L(\|\bar{w}^{\kappa}_{\text{act}_\kappa}\|, \|\bar{b}^{\kappa}_{\text{act}_\kappa}\|) \leq L_{\max} \\
& \forall \kappa = 1, \ldots, m \;.\; \beta(\|\bar{w}^{\kappa}_{\text{act}_\kappa}\|, \|\bar{b}^{\kappa}_{\text{act}_\kappa}\|) \leq \beta_{\max} \\
& \forall \kappa = 1, \ldots, m \;.\; L(\Xi^{(m)}_{N,M}) \leq L_{\max}
\end{aligned}$$

Note: the final collection of constraints on $L(\Xi^{(m)}_{N,M})$ is necessary to ensure that (4.21) is satisfied and Corollary 4.2 is applicable (equation (4.19) is satisfied by definition of $\beta_{\max}$).

4.5.2 Optimization Problem for Global Alteration (Activation)

If the optimization problem `Local` is feasible, then the local controller at $x_{\mathsf{c.e.}}$ can successfully be repaired, and the global activation of said controller can be considered. Since we are starting with a local linear function we want to be active at and around $x_{\mathsf{c.e.}}$, we can retain the definition of act_κ from the initialization of `Local`. Moreover, since Problem 4.1 preserves the selector matrices of the original TLL controller, we will define the selector indices, sel_κ, in terms of the activation pattern of the *original*, defective local linear controller (although this is not required by the repair choices we have made: other choices are possible).

Thus, in order to formulate an optimization problem for global alteration, we need to define constraints compatible with Proposition 4.4 based on the activation/selector indices described above. Part *(i)* of the conditions in Proposition 4.4 is unambiguous at this point: it says that the desired active local linear function, act_κ, must have the minimum output from among those functions selected by selector set s_{sel_κ}. Part *(ii)* of the conditions in Proposition 4.4 is ambiguous however: we only need to specify *one* local linear function from each of the *other* `min` groups to be "forced" lower than the desired active local linear function. In the face of this ambiguity, we select these functions using indices $\iota^\kappa_j : j \in \{1, \dots, M\}\backslash\{\text{act}_\kappa\}$ that are defined as follows:

$$\iota^\kappa_j \triangleq \arg\min_{i \in s^\kappa_j} [\![W^\kappa_\ell x_{\mathsf{c.e.}} + b^\kappa_\ell]\!]_i. \tag{4.28}$$

That is we form our global alteration constraint out of the non-active controllers which are have the *lowest outputs among their respective min groups.* We reason that these local linear functions will in some sense require the least alteration in order to satisfy Part *(ii)* of Proposition 4.4, which requires their outputs to be less than the local linear function that we have just repaired. Thus, we can formulate the global alteration optimization problem as follows:

$$\begin{aligned}
&\texttt{Global} : \min_{\overline{W}^\kappa_\ell, \bar{b}^\kappa_\ell} \sum_{\kappa=1}^{m} \|W^\kappa_\ell - \overline{W}_\ell\| + \|b^\kappa_\ell - \bar{b}_\ell\| \\
\text{s.t.} \quad & \forall \kappa = \{1, \dots, m\} \qquad\qquad . \; [\![W^\kappa_\ell]\!]_{\text{act}_\kappa, \cdot} = \bar{w}^\kappa_{\text{act}_\kappa} \\
& \forall \kappa = \{1, \dots, m\} \qquad\qquad . \; [\![b^\kappa_\ell]\!]_{\text{act}_\kappa, \cdot} = \bar{b}^\kappa_{\text{act}_\kappa} \\
& \forall \kappa = \{1, \dots, m\} \; \forall i \in s_{\text{sel}_\kappa} \; . \; \bar{w}^\kappa_{\text{act}_\kappa} x_{\mathsf{c.e.}} + \bar{b}^\kappa_{\text{act}_\kappa} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \leq [\![W^\kappa_\ell x_{\mathsf{c.e.}} + b^\kappa_\ell]\!]_i \\
& \forall \kappa = \{1, \dots, m\} \\
& \quad \forall j \in \{1, \dots, M\}\backslash\{\text{sel}_\kappa\} \; . \; [\![W^\kappa_\ell x_{\mathsf{c.e.}} + b^\kappa_\ell]\!]_{\iota^\kappa_j} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \leq \bar{w}^\kappa_{\text{act}_\kappa} x_{\mathsf{c.e.}} + \bar{b}^\kappa_{\text{act}_\kappa}
\end{aligned}$$

where of course $\bar{w}^\kappa_{\text{act}_\kappa}$ and $\bar{b}^\kappa_{\text{act}_\kappa}$ are the repaired local controller parameters obtained from the optimal solution of `Local`. Note that the first two sets of equality constraints merely ensure that `Global` does not alter these parameters.

4.5.3 Main Algorithm

A pseudo-code description of our main algorithm is shown in Algorithm 2, as `repairTLL`. It collects all of the initializations from Section 4.4, Subsection 4.5.1 and Subsection 4.5.2. Only the functions `FindActCntrl` and `FindActSlctr` encapsulate procedures defined in this paper; their implementation is nevertheless adequately described in Subsection 4.4.1.1 and Proposition 4.4, respectively. The correctness of `repairTLL` follows from the results in those sections.

4.6 Numerical Examples

We illustrate the results in this chapter on a four-wheel car described by the following model:

$$x(t+1) = \begin{bmatrix} x_1(t) + V\cos(x_3(t))\cdot t_s \\ x_2(t) + V\sin(x_3(t))\cdot t_s \\ x_3(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ t_s \end{bmatrix} v(t) \tag{4.29}$$

where the state $x(t) = [p_x(t)\ p_y(t)\ \Psi(t)]^T$ for vehicle position $(p_x\ p_y)$ and yaw angle Ψ, and the control input v is the vehicle yaw rate. The parameters are the translational speed of the vehicle, V (meters/sec); and the sampling period, t_s (sec). For the purposes of our experiments, we consider a compact workspace $X_{\mathsf{ws}} = [-3,3]\times[-4,4]\times[-\pi,\pi]$; a safe set of states $X_{\mathsf{safe}} = [-0.25, 0.25]\times[-0.75,-0.25]\times[-\frac{\pi}{8},\frac{\pi}{8}]$, which was verified using NNV [43] over 100 iterations; and an unsafe region X_{unsafe} specified by $[0\ 1\ 0]\cdot x > 3$. Furthermore, we consider model parameters: $V = 0.3$ m/s and $t_s = 0.01$ seconds.

All experiments were executed on an Intel Core i5 2.5-GHz processor with 8 GB of memory. We collected 1850 data points of state-action pairs from a PI Controller used to steer the car over X_{ws} while avoiding X_{unsafe}. Then, to exhibit a NN controller with a counterexample, a TLL NN with $N = 50$ and $M = 10$ was trained from a corrupted version of this data-set: we manually changed the control on 25 data points close to X_{unsafe} so that the car would steer into it. We simulated the trajectories of the car using this TLL NN controller for different x_0 and identified $x_{\mathsf{c.e.}} = [0\ 2.999\ 0.2]$ as a valid counterexample for safety after two time steps. Finally, to repair this *faulty* NN, we found all the required bounds for both system dynamics and NN parameters and a horizon of $T = 7$. We found the required safety constraints $\beta_{\max} = 0.0865$ and $L_{\max} = 1.4243$. Then, from $x_{\mathsf{c.e.}}$ we obtained the controller $K = [K_w\ K_b]$ where $K_w = [-0.1442,\ -0.5424,\ -0.425]$ and $K_b = [2.223]$.

Next, we ran our algorithm to repair the counterexample using CVX (convex solver). The result of the first optimization problem, `Local`, was the linear controller: $\bar{K}_w = [-0.0027\ -0.0487\ \ -0.0105]$ and $\bar{K}_b = [-9.7845]$; this optimization required a total execution time of 1.89 sec. The result of the second optimization problem, `Global` successfully activated the repaired controller, and had an optimal cost of 8.97; this optimization required a total execution time of 6.53 sec. We also compare the original TLL Norms $||W|| = 6.54$ and $||b|| = 5.6876$ with the repaired: $||\overline{W}|| = 11.029$ and $||\bar{b}|| = 5.687$.

Finally, we simulated the motion of the car using the repaired TLL NN controller for 50 steps. Shown in Fig. 1 are the state trajectories of both original *faulty* TLL controller and repaired TLL Controller starting from the $x_{\mathsf{c.e.}}$ In the latter the TLL controller met the safety specifications.

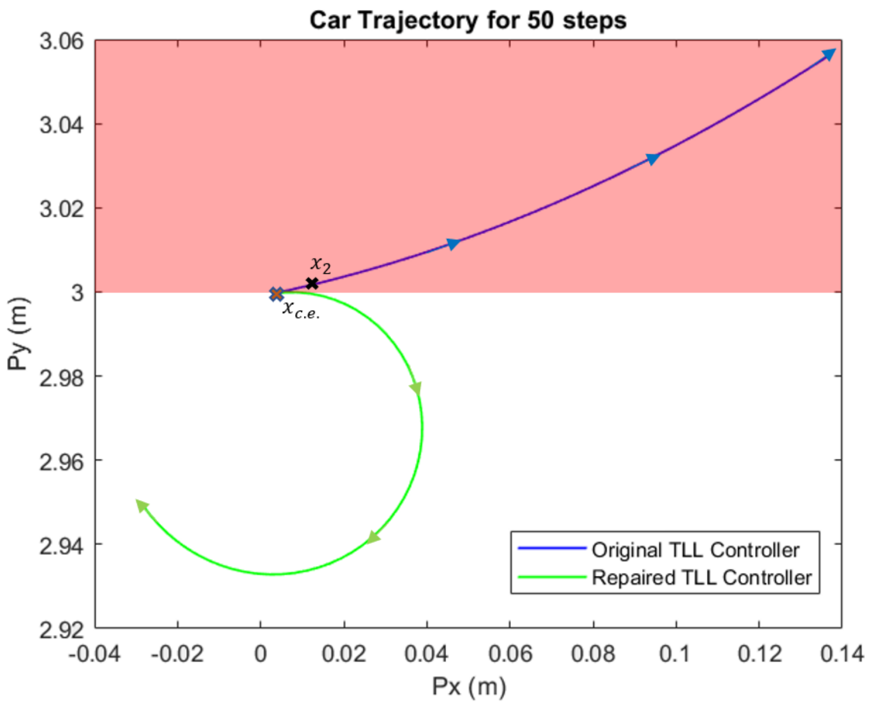


Figure 4.1: System starting from $x_{\mathsf{c.e.}}$ goes directly into X_{unsafe} and Repaired $x_{\mathsf{c.e.}}$ produces a safe trajectory. Red area is X_{unsafe}, Red Cross is $x_{\mathsf{c.e.}}$ and Black Cross shows state after 2 steps.

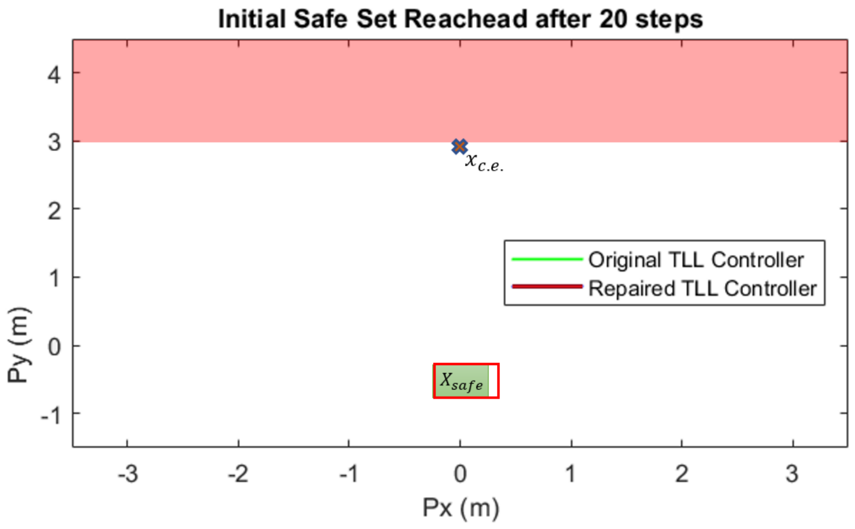


Figure 4.2: Initial Safe set before and after repair for 20 steps. Red area is X_{unsafe}; Red Cross is $x_{\mathsf{c.e.}}$; $X_{\mathsf{ws}} = [-3, 3] \times [-4, 4]$

```
input : $f, g$ system dynamics (4.1)
        $X_{\mathsf{ws}}$ workspace set
        $\Xi^{(m)}_{N,M}$ TLL controller to repair
        $T$ safety time horizon
        $X_{\mathsf{safe}}$ set of safe states under $\Xi^{(m)}_{N,M}$
        $x_{\mathsf{c.e.}}$ counterexample state
output: $\bar{\Xi}^{(m)}_{N,M}$ repaired TLL controller
1  function repairTLL($f$, $g$, $X_{\mathsf{ws}}$, $\Xi^{(m)}_{N,M}$, $T$, $X_{\mathsf{safe}}$, $x_{\mathsf{c.e.}}$)
2      gMaxSafe $\leftarrow \sup_{x_0 \in X_{\mathsf{safe}}} \|g(x_0)\|$
3      beta (w,b) := $\sup_{x_0 \in X_{\mathsf{safe}}} \|f(x_0) - x_0\|$
4          + gMaxSafe * w * ext($X_{\mathsf{ws}}$) + gMaxSafe * b
5      L (w,b) := $L_f$ + $L_g$ * w * $\sup_{x_0 \in X_{\mathsf{safe}}} \|x_0\|$
6          + w * gMaxSafe + $L_g$ * b
7      $\Omega_W \leftarrow \max_{w \in \cup_{\kappa=1}^{m} \{[\![W^\kappa_\ell]\!]_j | j=1,\dots,N\}} \|w\|$
8      $\Omega_b \leftarrow \max_{b \in \cup_{\kappa=1}^{m} \{[\![b^\kappa_\ell]\!]_j | j=1,\dots,N\}} \|b\|$
9      betaMax $\leftarrow$ beta( $\Omega_W$, $\Omega_b$)
10     dSafe $\leftarrow \inf_{\substack{x_s \in X_{\mathsf{safe}} \\ x_u \in X_{\mathsf{unsafe}}}} \|x_s - x_u\|$
11     Lmax $\leftarrow \inf \left\{ L' \middle| \text{ betaMax } * \sum_{k=0}^{T} L'^k = \text{dSafe} \right\}$
12     $\{\text{act}_\kappa\}_{\kappa=1}^m \leftarrow$ FindActCntrl($\Xi^{(m)}_{N,M}$, $x_{\mathsf{c.e.}}$)
13     $\{\text{sel}_\kappa\}_{\kappa=1}^m \leftarrow$ FindActSlctr($\Xi^{(m)}_{N,M}$, $x_{\mathsf{c.e.}}$)
14     Initialize(Local,{$f$, $g$, $\Xi^{(m)}_{N,M}$, $x_{\mathsf{c.e.}}$,L,Lmax, beta,betaMax,$\{act_\kappa\}_{\kappa=1}^m$, $X_{\mathsf{unsafe}}$})
15     sol $\leftarrow$ Solve(Local)
16     if not sol.feasible() then
17         return False
18     else
19         $\{(\text{w}^\kappa, \text{b}^\kappa)\}_{\kappa=1}^m \leftarrow$ sol.optimalValue()
20     end
21     for $\kappa$ in $1, \dots, m$ do
22         for $j$ in $\{1, \dots, M\} \backslash \{sel_\kappa\}$ do
23             $\iota^\kappa_j \leftarrow \arg\min_{i \in s_j} \|[\![W^\kappa_\ell x_{\mathsf{c.e.}} + b^\kappa_\ell]\!]_i\|$
24         end
25     end
26     Initialize(Global,{$f$, $g$, $\Xi^{(m)}_{N,M}$, $x_{\mathsf{c.e.}}$,L,Lmax,
           beta,betaMax,$\{act_\kappa\}_{\kappa=1}^m$, $\{\iota^\kappa_j\}_{\kappa,j}$, $\{(\text{w}^\kappa, \text{b}^\kappa)\}$})
27     sol $\leftarrow$ Solve(Global)
28     if not sol.feasible() then
29         return False
30     else
31         $\{(\text{W}^\kappa, \text{B}^\kappa)\}_{\kappa=1}^m \leftarrow$ sol.optimalValue()
32     end
33     return $\Xi^{(m)}_{N,M}$.setLinLayer($\{(\text{W}^\kappa, \text{B}^\kappa)\}_{\kappa=1}^m$)
34 end
```

Algorithm 2: repairTLL.

Bibliography

[1] International Verification of Neural Networks Competition 2020 (VNN-COMP'20). `https://sites.google.com/view/vnn20`.

[2] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[3] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification*, volume 12224 of *Lecture Notes in Computer Science*, pages 66–96. Springer International Publishing, 2020.

[4] R. W. Beard and T. W. Mclain. Small unmanned aircraft: Theory and practice. *Princeton University Press*, 2012.

[5] D. L. Berre and A. Parrain. The sat4j library. 7, 2010.

[6] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to algorithms. *The MIT Press*, 2003.

[8] G. Dong, J. Sun, J. Wang, X. Wang, and T. Dai. Towards repairing neural networks correctly, 2020. `http://arxiv.org/abs/2012.01872`.

[9] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[10] O. Faugeras and O. A. FAUGERAS. *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[11] J. Ferlez, M. Elnaggar, Y. Shoukry, and C. Fleming. Shieldnn: A provably safe nn filter for unsafe nn controllers. *arXiv preprint arXiv:2006.09564*, 2020.

[12] J. Ferlez, H. Khedr, and Y. Shoukry. Fast batllnn: Fast box analysis of two-level lattice neural networks. *arXiv preprint arXiv:2111.09293*, 2021.

[13] J. Ferlez and Y. Shoukry. AReN: Assured ReLU NN Architecture for Model Predictive Control of LTI Systems. In *Hybrid Systems: Computation and Control 2020 (HSCC'20)*. ACM, 2020.

[14] J. Ferlez and Y. Shoukry. Bounding the complexity of formally verifying neural networks: A geometric approach. *arXiv preprint arXiv:2012.11761*, 2020.

[15] J. Ferlez and Y. Shoukry. Bounding the Complexity of Formally Verifying Neural Networks: A Geometric Approach. In *2021 60th IEEE Conference on Decision and Control*, pages 5104–5109, 2021.

[16] J. Ferlez and Y. Shoukry. Polynomial-time reachability for lti systems with two-level lattice neural network controllers. *IEEE Control Systems Letters*, 2022.

[17] D. J. Fremont, J. Chiu, D. D. Margineantu, D. Osipychev, and S. A. Seshia. Formal analysis and redesign of a neural network-based aircraft taxiing system with verifai. In *International Conference on Computer Aided Verification*, pages 122–134. Springer, 2020.

[18] B. Goldberger, Y. Adi, J. Keshet, and G. Katz. Minimal modifications of deep neural networks using verification. *23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 73:260–278, 2020.

[19] P. Habeeb, N. Deka, D. D'Souza, K. Lodaya, and P. Prabhakar. Verification of camera-based autonomous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[20] R. Hartley and A. Zisserman. Multiple view geometry in computer vision. *Cambridge University Press*, 2003.

[21] C. Hsieh, K. Joshi, S. Misailovic, and S. Mitra. Verifying controllers with convolutional neural network-based perception: A case for intelligible, safe, and precise abstractions. *arXiv preprint arXiv:2111.05534*, 2021.

[22] C. Hsieh, Y. Li, D. Sun, K. Joshi, S. Misailovic, and S. Mitra. Verifying controllers with vision-based perception using safe approximate abstractions. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, 2022.

[23] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. 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)*, pages 120–129, 2018.

[24] N. Kallus and A. Zhou. Assessing disparate impact of personalized interventions: identifiability and bounds. *Advances in neural information processing systems*, 32, 2019.

[25] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 97–117. Springer International, 2017.

[26] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[27] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer. Verification of image-based neural network controllers using generative models. *arXiv preprint arXiv:2105.07091*, 2021.

[28] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer. Verification of image-based neural network controllers using generative models. *Journal of Aerospace Information Systems*, 19(9):574–584, 2022.

[29] S. Kauschke and J. Furnkranz. Batchwise patching of classifiers. *The 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[30] S. Kauschke and D. H. Lehmann. Towards neural network patching: Evaluating engagement-layers and patch-architectures, 2018. `http://arxiv.org/abs/1812.03468`.

[31] H. Khedr, J. Ferlez, and Y. Shoukry. PEREGRiNN: Penalized-Relaxation Greedy Neural Network Verifier. 2020.

[32] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An invitation to 3-d vision: from images to geometric models*, volume 26. Springer Science & Business Media, 2012.

[33] P.-J. Meyer. Reachability analysis of neural networks using mixed monotonicity. IEEE Control Systems Letters,vol. 6, pp. 3068-3073, 2022.

[34] T. Nagamine and N. Mesgarani. Understanding the representation and computation of multilayer perceptrons: A case study in speech recognition. In *International Conference on Machine Learning*, pages 2564–2573. PMLR, 2017.

[35] U. Santa Cruz and Y. Shoukry. Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing. NASA Formal Methods. NFM 2022. Lecture Notes in Computer Science, vol 13260. Springer, pp 213–230, 2022.

[36] S. Shoouri, S. Jalili, J. Xu, I. Gallagher, Y. Zhang, J. Wilhelm, J.-B. Jeannin, and N. Ozay. Falsification of a vision-based automatic landing system. In *AIAA Scitech 2021 Forum*, page 0998, 2021.

[37] M. Sotoudeh and A. V. Thakur. Correcting deep neural networks with small, generalizing patches. In *NeurIPS 2019 Workshop on Safety and Robustness in Decision Making*, 2019.

[38] X. Sun, H. Khedr, and Y. Shoukry. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 147–156, 2019.

[39] X. Sun, H. Khedr, and Y. Shoukry. Formal verification of neural network controlled autonomous systems. HSCC 19: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp 147–156, 2019.

[40] R. Szeliski. *Computer Vision - Algorithms and Applications, Second Edition.* Texts in Computer Science. Springer, 2022.

[41] R. Talak, L. Peng, and L. Carlone. Certifiable 3d object pose estimation: Foundations, learning models, and self-training, 2023.

[42] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Star-based reachability analysis of deep neural networks. In *Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings 3*, pages 670–686. Springer, 2019.

[43] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 3–17. Springer International Publishing, 2020.

[44] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I*, pages 3–17. Springer, 2020.

[45] Y.-S. Wang, L. Weng, and L. Daniel. Neural Network Control Policy Verification With Persistent Adversarial Perturbation. In *International Conference on Machine Learning*, pages 10050–10059. PMLR, 2020.

[46] M. Zamani. Control of cyber-physical systems using incremental properties of physical systems. *PhD Thesis*, 2012.

[47] M. Zamani, G. Pola, M. M. Jr, and P. Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control*, 57(7):1804–1809, 2012.