LEARNING AND VERIFICATION OF DYNAMICAL SYSTEMS WITH NEURAL NETWORK COMPONENTS

By

Diego Manzanas Lopez

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

August 12, 2022

Nashville, Tennessee

Approved:

Taylor Johnson Ph.D.

Radu Grosu Ph.D.

Janos Sztipanovits Ph.D.

Yuankai Huo Ph.D.

Gautam Biswas Ph.D.

Dedicated to my family, friends, and colleagues. Thank you very much for all your help and support!

ACKNOWLEDGMENTS

There have been numerous people who have been part of my PhD journey and who I would like to thank. First, I would like to express my deepest gratitude to my advisor and mentor, Dr. Taylor T. Johnson, for his continuous support and guidance for my research and professional career. I would also like to thank Dr. Radu Grosu, Dr. Janos Sztipanovits, Dr. Yuankai Huo, and Dr. Gautam Biswas for serving on my Ph.D. committee, and for the insightful discussions and feedback to this dissertation.

To my colleague and friend Patrick Musau, who has been there every step of the way. From learning together the basics of hybrid automata, machine learning, and many other once unfamiliar topics, to collaborating on many projects and completing the PhD, and, of course, for the many fun memories over the past 5 years. To my colleagues and collaborators Nathaniel Hamilton, Neelanjana Pal, Ayana Wild, Preston Robinette, Xiaodong Yang, Joel Rosenfeld, Stanley Bak and Kerianne Hobbs. Shoutout to Hoang-Dung Tran for all his help during my first few years of graduate school. I would also like to thank Timothy Darrah for giving me the opportunity of being part of his entrepreneurship adventures.

This endeavor would not have been possible without my family. Especially, I would like to express my gratitude to my parents, Jose Juan Manzanas and M^a Begoña Lopez for their constant love and support, and for providing me with the education and opportunities that made this journey possible. To my brother Sergio, for being the older brother I can look up to, for always being there for me when I needed it. And to my brother Carlos, for always pushing me to be a better tennis player, student, and person. Thanks for diving into this journey with me. Words cannot express my gratitude to Devika Harshan, thank you for all your support, patience, and entertainment during this journey. Thank you for standing by my side during all these years.

Lastly, I would be remiss in not mentioning several other people who have played a significant role in my life through their friendship. The tennis friends from Couder who helped me make the decision to come to the US, friends who make every reunion feel like we still see each other every day. PC friends, who welcomed me with open arms and always made me feel like home, and all the Nashville friends made along the way, for the spontaneous lake trips, game nights and all the fun times.

Acknowledgement of Support

This material is based upon work supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-22-1-0019, the National Science Foundation (NSF) under grant numbers 1918450, 1910017, and 2028001, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, nor NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

I.1 Motivation

Artificial Intelligence (AI) has proved to be very successful at certain tasks where clear boundaries and rules are defined. These mostly take the shape of Machine Learning (ML) or Deep Learning (DL) methods, and have proved to outperform some of the best humans in the world at such tasks, like Atari and DeepMind's AlphaGo [1]. These technologies have also been included in our everyday lives in various ways, such as recommendation systems in TV/movie applications like Netflix and Hulu, or as smart assistants like Alexa or Siri. The AI discipline and technologies are developed with the intention to match the learning and reasoning capabilities of the human brain, where the main successes in these areas are mostly achieved by the use of artificial neural networks (NN). The term neural network appeared several decades ago, motivated by the human brain structure in which information is passed from neuron to neuron. In this way, researchers in AI have made significant improvements in areas like image classification, object recognition, pattern recognition and more. These neural networks can have different shapes and architectures, and can be used in many tasks like regression and classification, but the underlying idea is the same: mimic a biological neural network in which information is transferred from neuron to neuron in which different processes may occur (activation functions).

Nevertheless, the success of these complex AI systems comes at a cost. NNs are considered to be opaque systems in nature, thus reasoning about their behavior is complicated. Being opaque systems means that there is a lack of transparency in the internal reasoning of the output decision-making, we can only reason about the input-output behavior. This type of behavior analysis limits the applications in which these NNs can be widely adopted. Cyber-Physical Systems (CPS) is an example in which the inclusion of NNs has been limited to non-critical tasks unless an exhaustive formal analysis has been conducted. A CPS is defined as the integration between a physical system that interacts with the real world and other complex computational processes that fully or partly command the behavior of these systems as they interact with the physical world. An example of a CPS is a modern car with some level of autonomy such as lane keeping technologies, in which the driver manually controls the car, but there are other behaviors like keeping the car in the lane that are controlled by a computer that processes the sensor information and sends steering commands to the car to keep it in the right lane. Other examples of CPSs can be home appliances with any computational processing units embedded in the system, like a dryer with a sensing unit to stop the unit when the clothes are dried.

For the former example of the partly autonomous car, due to the possible catastrophic consequences that an incorrect or unsafe behavior may lead to, exhaustive testing, validation and verification analysis is needed.

In many cases, due to the huge success in many areas such as image recognition, object detection or reinforcement learning, there are several tasks in autonomous CPS that are learned and commanded by deep learning models within the CPS, also referred to as Learning-Enabled Components (LEC). We refer to the systems as learning-enabled CPSs, which are the main systems that we investigate in this work. However, despite the great promises that learning-enabled CPSs hold to reinvent the way we move, the way we live, there are many challenges preventing an extensive adoption in safety-critical domains. Any system that integrates computational with physical capabilities, interacting with humans and the real world presents its own challenges, such as modelling and navigating the unstructured and dynamical environments these operate on, and the cross-domain knowledge required to design and integrate all the components of the systems, including but not limited to sensing, control, component communication, and actuators [2]. Theory and tools are needed to first design, analyze and verify components at various levels of the system, from both hardware and software components. Then, we need to analyze and understand the behavior and interactions of the whole system with all its subsystems, and finally guarantee that the overall system is correct and safe, meaning we need to verify that the implementation meets the design plan and its requirements. In addition to the CPS challenges, the inclusion of learning-enabled components adds a layer of complexity.

LECs have been proved to be susceptible to adversarial attacks. These adversarial attacks are generally small perturbations to the input of these components, which can drastically change its behavior [3]. Early studies have shown that small perturbation to images can cause a NN to misclassify them, i.e., a stop sign with a small perturbation is classified as a speed limit sign, which could have catastrophic consequences if deployed in a safety critical system like an autonomous or partially autonomous car. Motivated by the capabilities of these systems and the lack of proper understanding of the inside behavior, the area of neural network verification emerged a little less than a decade ago. The goal of these techniques are to prove whether the input-output neural network behavior is as originally designed. Depending on the task to be performed, the formal analysis of NNs can be computed in different ways and using different methods, although all of them reason about the input-output relationship.

However, analyzing the behavior of NNs in isolation is insufficient for its used in safety-critical systems. Therefore, as a logical and natural extension of these methods and their applications, several researchers began to investigate and formally analyze the full behavior of Learning-Enabled CPS. In general, this has been done by integrating neural network verification methods and hybrid system verification [4]. Hybrid system verification has existed for several decades now, and it is used to formally analyze the behavior of dynamical systems. These dynamical systems are typically defined by ordinary differential equations (ODEs)

which models the continuous dynamics of CPS, although more complex models can be used such as hybrid automata, which are a combination of discrete and continuous dynamics. Any of the functions describing the continuous evolution of the states of the CPS can be linear or nonlinear, and several techniques have been proposed to analyze them in both discrete and continuous manner. Despite many of the recent efforts, formal verification for hybrid automata and nonlinear systems remains a challenge in the form of scalability and conservativeness, where a tradeoff between the two exists [5].

Another challenge for the verification of learning-enabled CPS is to properly model the system dynamics. In order to provide mathematical guarantees on these physical systems, we need accurate model representations. Although in a perfect world we would be able to define the exact dynamics of every CPS using the laws of physics, this world is far from ideal and there are many instances in which we are not able to define the exact dynamics of the system to analyze. This is the case for many manufactured products, where a large percentage of parts are outsourced to other companies and the internal components of those parts are not always accessible to the user/manufacturer of the overall product. This is one example of many cases where the internal dynamics of a system may not be known, so we need to use data-driven techniques to model them. Depending on the complexity of the data, there exists methods to be used, including classic system identification techniques for linear and nonlinear systems, hybrid automata learning and deep learning methods [6, 7, 8, 9]. However, there are limitations on the type of dynamics that can be learned, i.e., learning hybrid automata with nonlinear continuous dynamics remains an open and challenging problem [8].

I.2 Research Challenges

Reasoning and understanding the behavior of learning-enabled CPSs requires a deep understanding about the individual components of the system, as well as the joint dynamics and interactions among such [10, 2]. Ideally, we would be able to define the joint dynamics of these systems based on the laws of physics based on our knowledge of each component and the interactions between them. However, there are many cases where this is not possible. In these scenarios, we need to make use of data-driven methods to identify the dynamics of these systems. While there are common and widely used techniques used to learn linear ODEs from data, and there are several for nonlinear systems [9], the methods for hybrid systems are more limited due to the complexity of these that combine continuous and discrete behavior [8]. There are also other recent methods from the deep learning area such as RNNs, neural ODEs, or the combination of both that have the capability to learn such dynamics [6, 7, 11, 12].

The second step to reason and analyze their behavior is with the use of validation and verification methods. In safety-critical domains, we need to have guarantees that the system will operate safely and correctly according to its design and specifications. The best methods to increase the trust about these systems is with

the use of formal verification methods. However, these often suffer from scalability issues, specially when dealing with complex systems with high dimensions and many parameters. In addition, when these systems are controlled with neural network controllers or LECs, the verification complexity increases even further due to the opaque nature of these neural network controllers [13]. In addition, for some of these deep learning models like neural ODEs, there are no empirical methods to verify them, becoming infeasible to use in safety-critical applications.

Spurred by existing challenges and limitations, this document focuses on addressing the following questions:

- Are current verification approaches suitable to formally verify complex learning-enabled CPSs in safety-critical domains? If not, can we extend these or develop new methods to verify them?
- Neural ordinary differential equations present promising advantages over existing classical system identification and deep learning methods. Are these models robust? Can we provide formal guarantees on their behavior?
- How can we learn the complex dynamics of CPSs that present discrete and nonlinear continuous behavior? Are existing methods enough to approximate these systems?

I.3 Research Contributions

Motivated by the existing challenges, the proposed research contributions are presented and discussed in detail in Chapters III to VII, and can be summarized as follows:

- We begin by examining two case studies of neural network control systems (NNCS) in the area of autonomous vehicles with the Neural Network Verification (NNV) tool [14]. These two case studies aim to extend existing methods in NNV with increased complexities in the NNCS.
- The first case study formally verifies the safety of an unmanned underwater vehicle, where the main challenge is to verify the 3 neural networks that the control system is composed of. We extend the NNV tool to handle NNCS with multiple neural networks as part of the system [15]
- The second case evaluates the safety of an unmanned aircraft using the neural network compression of the Advisory Collision Avoidance System X for unmanned aircraft (ACAS Xu), where the methods in NNV are extended to handle the switching behavior of the classification-based controllers of the ACAS Xu [16]. This study shows that it is possible to do a comprehensive safety verification analysis of a complex air collision advisory system for aircraft travelling in straight, co-altitude paths.

- Then, we present our work on the verification of neural ordinary differential equations (ODEs). We introduce a general class of neural ODEs (GNODEs), then describe the methods implemented in our framework, and finally evaluate these in a set of benchmarks from the areas of dynamical systems, control systems and image recognition.

- Finally, we introduce a nonlinear hybrid automata learning framework that makes use of an occupational kernel system ID and neural ordinary differential equations to learn the continuous dynamics of hybrid systems. This framework improves state-of-the-art tool performance due to the incorporation of continuous nonlinear dynamics to describe the continuous evolution of the system dynamics.

I.4 Organization

The remainder of this dissertation is organized as follows. Chapter II presents a summary of the recent research in the areas of hybrid automata learning, a discussion of verification approaches for neural network control systems (NNCS), including a description of state-of-the-art tools, and an introduction to neural ODEs, including its different architectures and learning techniques with its applications, and the existing works on verification of neural ODEs. Chapter III introduces the first case study of the NNCS verification work: an experimental evaluation of formal verification of an underwater vehicle system with three neural networks. Chapter IV presents another experimental evaluation of a NNCS verification of unmanned aircraft with switching classification-based controllers. Chapter V describes the framework and methods for the verification of neural ordinary differential equations and its experimental evaluation. Chapter VI presents the hybrid automata learning framework that makes use of two nonlinear system identification routines to learn the continuous dynamics of the hybrid systems, as well an evaluation and comparison to a state-of-the-art approach. Chapter VII ends with the concluding remarks, remaining challenges and future work, followed by the author's list of publications and presentations in Chapter VIII.

I.5 Copyright Acknowledgements

We include the following required copyright statements for permission to reprint portions of [15] and [16].

- For portions of [15] reproduced in this dissertation, we acknowledge the IEEE copyright: ©2020 IEEE. Reprinted, with permission, from Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, Hoang-Dung Tran, and Taylor T. Johnson, "Case Study: Safety Verification of an Unmanned Underwater Vehicle", 2020 IEEE Security and Privacy Workshops (SPW), 2020, pp. 189-195, May 2020.

- For portions of [16] reproduced in this dissertation, we acknowledge the AIAA copyright: ©2020 AIAA. Reprinted, with permission, from Diego Manzanas Lopez, Taylor Johnson, Hoang-Dung Tran,

Stanley Bak, Xin Chen, and Kerianne L. Hobbs, ”Verification of Neural Network Compression of ACAS Xu Lookup Tables with Star Set Reachability,” AIAA 2021-0995. AIAA Scitech 2021 Forum, January 2021.

CHAPTER II

Related Work

The research area in the intersection of CPS, deep learning, and formal methods is very large, so we mainly focus on describing the state-of-the-art methods and tools in three subareas more closely related to our proposed work: hybrid automata learning, neural network control system (NNCS) verification, and learning and verification of neural ODEs.

II.1 Hybrid Automata Learning

II.1.1 Preliminaries

A hybrid automaton is a model that presents both continuous and discrete behaviors in its dynamics. The continuous dynamics are typically defined by Ordinary Differential Equations (ODEs) and describe the continuous evolution of the state variables of the system, while the discrete dynamics define the jumps or transitions between different ODE modes that describe different continuous evolution of the state variables of the model. This type of models is also referred to as hybrid system, and to properly understand the state-of-the-art methods to learn hybrid systems, we begin by formally describing a hybrid automaton in Definition 1.

Definition 1 (Hybrid Automaton) *A hybrid automaton is defined as a tuple HA = (Z, z_0, X, x_0, U, inv, T, g, h, f) where:*

- $Z = \{z_1, z_2, ..., z_N\}$ *is the finite set of locations, with an initial location* $z_0 \in Z$.
- $X \subseteq \mathbb{R}^n$ *is the continuous state space with a set of initial continuous states* x_0 *such that* $x_0 \subseteq inv(z_0)$.
- $U \subseteq \mathbb{R}^n$ *is the continuous input space.*
- $inv : Z \rightarrow 2^X$ *is a mapping that assigns an invariant* $inv(z) \subseteq X$ *to each location z. An invariant is a property of each location that must be satisfied by all the continuous states for a given location. A state* $(z, x) \models inv(z)$ *iff* $x \in inv(z)$.
- $T \subseteq Z \times Z$ *is the set of discrete transitions between locations. A transition from* $z_i \in Z$ *to* $z_j \in Z$ *is defined as* (z_i, z_j).
- $g : T \rightarrow 2^X$ *is a guard function that associates a guard set* $g((z_i, z_j))$ *for each transition from* z_i *to* z_j, *where* $g((z_i, z_j)) \cap inv(z_i) \neq \emptyset$.

- *$h : T \times X \rightarrow X$ is the reset function that returns the next continuous state when a transition is taken. The reset function of a transition $g((z_i, z_j))$ is denoted as $h((z_i, z_j))$.*

- *$f : Z \times X \times U \rightarrow \mathbb{R}^n$ is the set of ordinary differential equations (ODEs) that define the continuous dynamics for each location $z \in Z$ over the continuous state variables $x \in X$, and it is denoted as $\dot{x} = f(z, x, u)$.*

The invariants $inv(z)$, guard sets $g((z_i, z_j))$ and reset functions $h((z_i, z_j))$ are modeled as linear functions and the former two are defined as bounds over the continuous state variables. An *execution* of a hybrid automaton model *HA* is an alternating sequence of continuous flow trajectories, which evolve as defined by each location's continuous dynamics and are bounded by the *invariants*, and discrete transitions between these locations defined by the *guard* and *reset* functions.

II.1.2 Learning approaches

Having defined what a hybrid system is, we can have a better understanding on how these systems work and the challenges and complexities. While there have been more works for learning purely continuous systems or purely discrete systems, hybrid systems have not been exhaustively studied despite recent efforts [17, 18, 19, 8, 20, 21, 22, 23]. In this category of hybrid system we include some variations such as timed and switched systems, that while simplifying some of the aspects of hybrid automata, many of the learning aspects are very similar and can be applied to general hybrid systems. Some of these studies have focused on the learning of Switched affine AutoRegressive eXogeneous (SARX) and PieceWise affine ARX (PWARX) models [24, 25, 26]. There are mainly three classes of approaches to learn such models, including algebraic based [27, 28, 29, 30, 31], clustering based [32, 33, 34, 35, 36], and optimization based [37, 38, 39, 40, 41]. Most of these works focus on purely identifying the dynamics of the system modes, but there are very few that study the learning of the hybrid system as a whole, including the overall number of modes, transitions, and guards.

In the past few years, there have been multiple attempt at studying hybrid automata inference. Niggeman et al. [23] present a technique called HyBUTLA, which was the first method to solve the hybrid automata learning problem at scalable precision. To learn the dynamics, they estimate each mode using linear regression or neural networks. After the identification has been done, the modes are evaluated based on the probability of staying in the state similarities and merged in a bottom-up manner. The transition between modes are calculated using transition probabilities, timing constraints and events. Grosu et al. [42] proposed a method to learn a variation of HA called Cycle-Linear Hybrid Automata (CLHA). The goal is to learn the dynamics of cells using electric signals known as the action potential (AP). They make use of AP properties

to identify the modes and the transitions of the corresponding HA. Then, the linear dynamics for each mode are learned using a modified Prony's method. Finally, the modified Prony's method is applied again to the dynamics and transitions to learn a function that adjusts the mode dynamics and switching logic on a per-cycle basis.

Medhat et al. [17] propose a learning framework using LearnLib [43] to learn hybrid automata with input/output traces. First, they cluster the traces based on input/output events and to these, linear regression is applied to learn the mode dynamics. Summerville et al. [44] introduce the Causal Hybrid Automata Recovery via Dynamic Analysis (CHARDA). This technique has two clearly differentiated steps: mode identification and causal guard learning. First, the modes are identified using a dynamic programming approach to segment the data into traces and then compute a cost function with a penalty criterion based on a template set of linear models to select the optimal model with the minimum trace error. Then, it learns causal guard conditions for the transitions between the dynamic modes using Normalized Pointwise Mutual Information (NPMI). A similar work to CHARDA is HyMn [45], although there are some key differences. The main idea is the same, to learn a linear hybrid system from input/output traces, although HyMn does not require any prior knowledge in the dynamics (no templates necessary). HyMn Sarkar et al. [46] address the learning of stochastic switched linear systems from noisy input/output data. The focus of this work is to learn lower-order approximations of the system by using Hankel-like matrices to construct data and use Singular Value Decomposition (SVD) truncation to calculate the approximated dynamics.

Garcia Soto et al. [18] propose two algorithms to apply membership-based synthesis to learn linear hybrid automata with nondeterministic guards and invariants. A main difference with other methods described is that these algorithms do not require a template for the dynamics, and work with an arbitrary number of modes. The first approach minimizes the number of modes encoding the synthesis problem as a logical formula via linear arithmetic, and then it's solved with an SMT solver. However, this is only feasible for a limited number of datasets. The second method targets this drawback by iteratively extending an initial synthesized hybrid automaton based on the processing of new training data. Thus, this method is applicable to online estimation as well as synthesis-in-the-loop applications, although it is not a minimal model. Later on, these methods are extended to consider affine dynamics following a principled search algorithm for the automation modification [19]. This method provides closeness guarantees with respect to precision parameters. Bernhard et al. [47] introduces a method to learn hybrid automata combining automata learning techniques with Model-Based Testing (MBT) to generate proper training data and then use machine learning to learn the dynamical behavior of the hybrid system. This method shows a significant improvement with respect to previous methods without requiring any a priori knowledge on the dynamics of the system. However, the use of LSTM-RNNs increases the complexity of the system and reduces the number of formal verification techniques that can analyze these

systems. Yang et al. [8] recently introduced a framework (HAutLearn) to identify and validate affine hybrid automata from input/output traces. This framework approximates the dynamics of each mode using linear ODEs for the segmented traces previously clustered. Then, a modified subspace-clustering technique is used to compute the linear inequalities of the guard conditions for each transition between the clustered segments. Finally, a prefix tree acceptor method is applied to reduce the number of modes and transitions by merging the similar modes and generate the resulting hybrid automaton.

II.2 Neural Network Control System Verification

Verification of CPS with learning-enabled components has been a hot topic in recent years due to the application and capabilities of machine learning applied to CPS and its promising results. However, verification is needed for this to be applicable to safety-critical systems. Several methods have been proposed to verify the safety of these feedback neural network control systems [14, 48, 49, 50, 51, 49, 52, 53, 54, 55, 56, 57, 58, 59], which in most cases consists on two key steps: neural network verification and hybrid system verification. The first one typically used for the controller, the second one applied to the plant. For this work, we will limit our study to evaluate and compare to methods and tools that are applicable to NNCS. For a more detailed discussion on neural network verification and hybrid system verification methods individually, we refer the reader to [60, 4] and [5] respectively. We introduce the different approaches and tools including NNV, Verisig, Sherlock, SMC, ReachNN and VenMAS. For an in-depth discussion of verification of NNCS in terms of verification approaches, we refer the reader to [61], although some of the most recent tools discussed were not included.

II.2.1 State-of-the-art Approaches

NNV

NNV is a neural network verification toolbox focused on reachability analysis techniques of neural networks and NNCS [14]. The first techniques implemented used polyhedron as the reach set representation, and then moved onto zonotope and star set reachability analysis. The early polyhedron-based approach for ReLU feedforward neural networks was extended for safety verification of a NNCS in [54] where a ReLU network controller controls a discrete linear switching plant model. This approach computes the polyhedron-based reachable set of the neural network controller at every control step. This first approach suffered from scalability problems and could only be used with a small neural network and a low-dimensional plant. This is due to the conservativeness on the computation of the convex hull of the reach sets, so the over-approximation error may accumulate quickly at every control period. Improving this approach, Weiming et al. proposed a new simulation-guided approach for safety verification of an NNCS [62]. The main idea of this approach is

to combine interval arithmetic and simulation-guided input set partition to estimate the neural network's output ranges, which are used as inputs for the plant model reachability. The presented results demonstrate the capabilities of this approach to efficiently verify the safety of a neural network-based adaptive cruise control system. Additionally, this approach is much less expensive than the maximum sensitivity based approach [63].

The zonotope methods for NNCS included in this tool are a combination of zonotope reachability for neural network [64] and CORA [65].

Later on, these methods have been extended to the use of star sets to improve the scalability and reduce the over-approximation errors of the polyhedra [14]. This approach has been extended to other activation functions such as LeakyReLU, tanh and sigmoid, which takes the same approach as the ReLU reachability analysis, but in the case of the nonlinear activation functions only the over-approximate reach set can be computed [14].

Verisig

Verisig [50] converts neural networks with smooth activation functions (tanh/sigmoid) to hybrid automata and encodes the full NNCS into Flow* [66]. This is done by exploiting the properties of sigmoid and tanh, which are solutions to a quadratic differential equation, and therefore the neural network equations may be expressed as an equivalent hybrid system. To improve the existing solutions using Flow*, several extensions have been implemented into this tool, including Taylor model preconditioning, shrink wrapping and a parallelizable regime in C++. Despite the use and extensions made on Flow* to optimize solving NNCS, this approach could be also encoded in other hybrid system reachability tools such as CORA [65] or SpaceEx [67].

VenMAS

VenMAS [56] is a tool for verification of NNCS. This tool encodes the verification problem as a Mixed-Integer Linear Programming (MILP) problem and uses Gurobi to solve it [68]. It presents the problem as a combination of agents and the environment that the agents interact with. A main difference with the previously mentioned reachability tools is that it needs a property to be verified to compute results, like any other SAT or MILP optimization-based tool.Verification tools based on reachability analysis are able to produce all the possible reach sets of the system (given a set of initial conditions) and then reason over one or multiple properties as desired. VenMAS supports only piecewise-linear activation functions for the controllers, and the property to verify needs to be expressed as a bounded Computational Tree Logic (CTL). Similar to other frameworks like NNV or Verisig, they also extend their work to parallel computing to optimize the problem solving.

Sherlock

Sherlock was developed as a neural network verification tool [69] and then extended to formally analyze NNCS [49]. For the neural network analysis, it uses an interleaving of gradient based local search technique and MILP to solve the global optimization problem for the range analysis of NNs. To extend it to NNCS, it adds a generation of polynomial rules to characterize the local behavior of the neural network which then are combined with the computed Taylor models of the plant using Flow* which allows for a better error analysis technique to reduce the wrapping effect in flowpipe computation. In [49], the authors show the efficiency of this technique over the simple combination of Flow* + the original NN range analysis in Sherlock.

OVERT

OVERT [57] is a NNCS verification tool designed to analyze discrete-time plants with ReLU and linear neural network controllers. OVERT presents the verification problem using an MIP encoding with the Gurobi solver [68]. The MIP encoding is generated by over-approximating the nonlinear dynamics of the plant using piecewise linear functions (not necessary if dynamics are already piecewise linear). These over-approximations are generated using relational abstractions, which makes the method sound, module finite precision arithmetic. Technically, the way the verification problem is set, it may be solved by any constraint solving tool capable of reasoning over piecewise-linear functions.

ReachNN*

ReachNN* [70, 58] is a NNCS verification tool that, similar to other like Sherlock or Verisig, integrates Flow* within their framework to compute the reachable set of the plant. In terms of the neural network analysis, ReachNN* uses Bernstein polynomials to approximate any Lipschitz-continuous NN with provable error bounds. The main advantage of using Bernstein polynomials is: 1) They can be interpreted and used by hybrid systems reachability tools like Flow* and 2) benefiting from the fact that Bernstein polynomials are universal approximators, the neural network verification is not limited to a specific set of activation functions. It can handle piecewise-linear as well as nonlinear functions. Based on the fact that this method relies on the partition of the state space to propagate the reach sets, this is further optimized by the use of GPU-based parallel computing.

JuliaReach

JuliaReach [71] is an open-source software suite for reachability analysis of dynamical systems and feed-forward neural networks with piecewise-linear layers. This framework is a collection of libraries written in Julia, with the latest addition of a NNCS library named NeuralNetworkAnalysis.jl. For the plant verification,

they implement the sound *TMJets* algorithm, based on interval arithmetic and Taylor models. This is a variant of the jet transportation using a Taylor polynomial with interval coefficients that allows the user to tweak the reachability algorithm based on a set of main parameters that adjust the tolerance and the order of the Taylor expansion in the reachability computation of the plant states. Other reachability methods such those available in SpaceEx [67] or Bak et al. [72] for LTI systems are also integrated into the framework. For the neural network analysis, they implement the zonotope-based methods also available in NNV, originally introduced by Gehr et al. [73]. The main benefit of this framework is that it is written in Julia, which combines the rapid-prototyping with high performance.

POLAR

POLAR [59] is a reachability analysis framework for NNCS based on polynomial arithmetic. Unlike existing approaches that use standard Taylor models, this framework makes use of a combination of Bernstein polynomials and Taylor model arithmetic to compute the reachable sets of the neural networks. For the plant analysis, POLAR makes use of Taylor Model (TM) flowpipe construction techniques for computing the reachable sets of ODEs described in [66, 74]. Their approach combining TM arithmetic and Bernstein polynomials is able to overcome the main drawbacks of TM arithmetic, significantly improving the accuracy and overapproximation errors in the computation of the reachable sets of NNCSs. In addition to this combination, this method keeps the TM remainders symbolic at every step under the linear mappings when computing the output set of the neural network controller.

II.2.2 Tool Comparison

During the past few years, we have hosted a friendly competition to evaluate and compare the state-of-the-art NNCS tools: *Artificial Intelligence and Neural Network Control Systems (AINNCS)* at the Applied Verification for Continuous and Hybrid Systems (ARCH). Several of the tools mentioned earlier have participated in the past 3 years where continuous and discrete models, and linear and nonlinear dynamical plants are evaluated as well as controllers with nonlinear and piecewise-linear activation functions to comprehensively compare the benefits and disadvantages from each method and tool with one another. However, this is a friendly competition during the first few years of existence, so the benchmarks studied are simply a baseline to compare, and these include an adaptive cruise control (ACC) [55], the VerticalCAS [75] and a TORA [49] among others. Despite the growing efforts in the past decade, scientifically speaking this is a relatively new area with lots of challenges ahead. It is observed during all the iterations that there is not a single tool able to outperform the other tools in all scenarios. Only NNV, Juliareach and POLAR are able to analyze all the benchmarks, including any type of dynamics and any controller with ReLU, leakyReLU, tanh, sigmoid and more activation

functions. VenMAS and OVERT were designed to formally verify only discrete-time systems, so any NNCS with continuous-time dynamics needs to be discretized prior to its verification. ReachNN* does not support controllers with multiple outputs, Sherlock dos not support neural network controllers with smooth activation functions, and Verisig only supports smooth activation functions (tanh/sigmoid) Although JuliaReach, POLAR, and NNV are able to support both piecewise-linear and smooth activation functions, this does not mean they are always better, as they are not always optimized to solve every reachability problem [13, 76, 77]. The reason that these tools do not support all the benchmarks is because they are optimized to successfully verify a specific problem in this area, and they should be used for those specific problems. The AINNCS ARCH competition present a great report to correctly choose the tools to use depending on the problem at hand [13].

Other Works: Case Studies

To extend this comprehensive evaluation of the tools and evaluate them with more complex systems that closely resemble those of real world applications, some examples have been presented, such as the safety verification of an autonomous car with LIDAR using Verisig [78]. This is one of the areas we target and extend in this work by formally verifying two CPS with varying complexities for unmanned underwater and aerial vehicles in Chapters III and IV. Using different methods to those previously described, the work by Sun et al. [52] was the first approach to handle LIDAR inputs processed by the NN controller. The approach is developed around the construction of a finite-state abstraction of the NNCS, which is then formally verified given a set of initial conditions. This method only works with ReLU/linear NN controllers and discrete-time linear plants. The abstraction constructed of this type of NNCS is then encoded as a Satisfiability Modulo Convex (SMC) formula and fed into an SMC solver, which reasons over the states of the system using a combination of a convex programming solver and a Boolean satisfiability solver. There have also been several works that verify or partially verify a NNCS of unmanned aerial vehicles with either horizontal or vertical commands [79, 80, 81, 82] use a combination of symbolic interval analysis with Reluval as the NN verification tool [83] with other hybrid system verification methods to verify these autonomous systems.

II.3 Neural ODEs

Neural ODEs were first introduced in 2018 by Chen et al. [6] in which they introduce the foundations and some applications of neural ODEs. Based on existing architectures such as recurrent neural networks (RNNs) and residual networks, which based their functions as a composed sequence of discrete transformations to a hidden state, the continuous-depth generalization of such architectures emerged as neural ODEs. In general, and including several of the new variants and improved neural ODE models, the advantages of Neural ODEs are memory efficiency, since they can reduce the number of parameters in the overall system with the use of

the ODE layers, improvement in the performance on time-series prediction over neural network models such as LSTMs and RNNs in terms of learning dynamical models with irregular sampling methods. However, there are also some drawbacks like learning stiff ODEs, which significantly increases the number of function evaluations (NFEs) and therefore the execution times, reducing the effect of the memory efficiency. Due to the potential computation benefits of neural ODEs, there have been numerous efforts in the area in the last couple of years.

II.3.1 Development of Neural ODEs

Since originally introduced, several variants and improvements have been proposed, such as Dupont et al. [84] that proposes a more expressive neural ODE by augmenting the state space of the neural ODE to allow the flow trajectories to cross and learn more complex functions that neural ODEs cannot represent. This is achieved by adding a vector of 0s in the input for each extra dimension, in order to increase the state-space dimensionality. These are called augmented neural ODEs, and some extensions to these models as [85] and [86] have been later proposed. Norcliffe et al. [87] introduced second order neural ODEs (SONODE) as an extension to augmented neural ODEs. They show how the adjoint methods can be extended to deal with these type of systems as well as other higher dimensional neural ODEs. They show via an empirical evaluation on synthetic and real dynamical systems that SONODEs generally result in faster training and better performance. Choromanski et al. [88] present a new paradigm for neural ODEs in which time-dependent parameters of the main flow evolve according to a matrix flow. Their main contribution is the use of learned isospectral flows as continuous systems as stabilizers for training neural ODEs. This leads to more efficient and stable training, which it is demonstrated by experiments run on popular RL environments (OpenAI Gym) [89] FFJORD [90] presents a similar idea to neural ODEs in the form of a reversible generative model with continuous-time dynamics. This model maps points from a simple distribution to a complex distribution, which improves the likelihood estimation $O(D^2)$ to O(D) time cost by using the Jacobian determinants. The dynamics of FFJORD are defined by a neural network and incorporate a time variable to each layer as an input. It improves previous methods such as Glow and Real NVP by adding flexibility to the architecture of the model, which improves the performance on density estimation and variational inference. Finlay et al. [91] define a new method to improve training for Neural ODEs over the common numerical solvers used. This work brings neural ODEs closer to practical relevance in large-scale applications by a combination of optimal transport and stability regularization, which lead neural ODEs to prefer simpler dynamics to solve the problem at hand without loss in performance. In general, these regularized simpler dynamics leads to a reduction in training time and NFEs. Zhang et al. [86] presents an extension to the method proposed by [92] which avoids the possibility of encountering incorrect gradients due to numerical instability and also

the inconsistencies that relate to optimizing infinite dimensional operators. However, the proposed ANODE presents some limitations in the sense that using more time steps or different discretization schemes do not affect model's generation performance. The proposed ANODEV2 is a more general neural ODE framework that addresses this problem. This framework also allows the evolution of both weights and activations by a coupled system of ODEs. Rubanova et al. [12] present a new type of neural ODE named as Latent ODE-RNNs, which improve standard RNNs by generalizing these to have continuous-time hidden dynamics defined by ODEs. They use this network to replace the recognition network recently proposed Latent ODE model [6]. Both of these can naturally handle arbitrary time gaps between observations, and can explicitly model the probability of observation times using Poisson processes. Comparison shows these models outperform RNNs on irregularly-sampled data, although the Latent ODE-RNNs are the best performing models for these type of problems. Kidger et al. [93] based their approach on controlled differential equations (neural CDEs) to solve a fundamental issue in which the solution to an ODE is determined by its initial condition. These models are a different extension to Latent ODEs, and they outperform Latent ODE-RNNs on several aspects such as training time and performance based on the ODE solver and procedure used. For neural CDEs, the steps of its numerical solver can be made across observation, while the ODE-RNN solver must solve each observation. Morril et al. [94] extend neural CDEs to neural rough differential equations (neural RDEs), which allow for a more general class of driving signals for processing the inputs.

II.3.2 Analysis of Neural ODEs

Many of the methods explained aim to develop new model architectures, learning algorithms or finding new applications. However, very little work has done to formally analyze neural ODEs, even treating neural ODEs as general input-output models. In this area, Massaroli et al. [95] aim to *dissect* these models by a detailed reasoning behind the parameters and core components used in these models, backed with experiments including depth-variance, augmented vs. non-augmented neural ODEs and data controlled neural ODEs. They present an analysis on how different parameters and hyperparameters of the model affect the learning and performance. Carrara et al. [96] and Yan et al. [97] look into the robustness of neural ODEs. The former one analyzes the robustness of image classifiers implemented as neural ODE and compares to standard CNNs, which shows Neural ODEs to be more robust than state-of-the-art residual models. In addition, thanks to the continuity of the hidden state, it is possible to track the perturbation injected by adversarial inputs and detect which part of the internal dynamics are most responsible for a misclassification. Experiments are conducted using the popular MNIST [98] and CIFAR-10 [99] datasets. The latter one explores the robustness properties of neural ODEs in general, both empirically and theoretically. In addition to this, they also introduce a new time-invariant steady neural ODE (TisODE), which further enhances the robustness of vanilla neural ODEs

by regularizing the flow on perturbed data via the time-invariant property and the imposition of a steady-state constraint. Their results are conducted using MNIST [98], SVHN [100] and a subset of ImageNet [101] (only 10 categories, resized images). Gruenbacher et al. [102] are the first ones to consider formal verification of Neural ODEs in a theoretical way, no experiments were presented. Their main contributions are the introduction of Stochastic Lagrangian Reachability (SLR) [103], which is an abstraction-based technique for computing an over-approximation set of states over a finite-time horizon, and provide stochastic guarantees in the form of confidence intervals for the reach sets. One key element of their methods is to avoid the accumulation of over-approximation errors by performing local optimization steps to expand safe regions instead of repeatedly propagate them as in standard deterministic reachability methods. These fast optimizations are achieved by a forward-mode adjoint sensitivity method to compute gradients without the need for backpropagation. In a follow-up work, this approach was improved and implemented in a verification tool named GoTube [104]. This tool is able to compute the reachable sets of neural ODEs for longer time horizons than state-of-the-art hybrid system verification tools, as demonstrated in [104]. However, their method is only applicable to neural ODEs with nonlinear and smooth activation functions, and there are no formal guarantees on the derived results, only stochastic bounds are provided.

II.3.3 Neural ODEs as dynamical systems

Another clear extension path of the use of neural ODEs is to apply them to other dynamical systems, like hybrid automata and stochastic dynamical systems. There have been some works like Graph Neural ODEs [105], Latent Neural ODEs from trace segments [106], Neural Event ODEs [11], Neural Jump Stochastic DEs (NJSDEs) [7], and Neural Hybrid Automata with probabilistic transitions [107]. Poli et al. first introduce Graph Neural ODEs (GDEs), which was their first a work to include discrete and continuous dynamics, similar to hybrid systems [105]. The main idea of this work is to represent the dynamics of graph neural networks (GNNs) using differential equations, these are the continuous-depth counterpart to GNNs. Later on, they extend this work to learn hybrid dynamics using multiple modes and stochastic transitions [107]. The proposed model (NHA) is a model to learn Stochastic Hybrid Systems (SHSs), comprised of a dynamics module that represents the continuous-time dynamics of the multiple modes, a discrete latent space that identifies in which mode in the hybrid system the model is, and the event module, which learns when event happens and how the transitions occur. The learning procedure for the overall system is similar to that of hybrid automata in general, where first the data in clustered into trace segments and then the different methods are applied to separately learn the dynamics, events, and transitions. Similarly to NHAs is the work by Jia et al. (NJSDEs), which can be explained as a SHS with only one mode and stochastic *jumps* or self-transitions, in which the continuous dynamics change according to the dynamics of the transitions in a discrete manner

[7]. Neural Event ODEs are very similar to this work, but with a big difference in the way transitions can happen [11]. While these systems may only have one mode, same as NJSDEs, the events and transitions are deterministic instead. Shi et al. [106] present a model called Latent Segmented ODE (LatSegODE), which is an extension of latent ODEs to include discrete dynamics into the system. It is very similar to neural event ODEs, but the main difference is how the continuous dynamics are modeled, where neural event ODEs uses neural odes, Shi et al. propose a model with latent ODEs.

CHAPTER III

Neural Network Closed Loop Verification of an Unmanned Underwater Vehicle

This chapter is adapted from the material presented in [15]

In this chapter, we evaluate the safety of a neural network controller that seeks to ensure that an Unmanned Underwater Vehicle (UUV) does not collide with a static object in its path. To achieve this, we utilize methods that can determine the output reachable set of all the UUV's components through the use of star-sets. The star-set is a computationally efficient set representation adept at characterizing large input spaces. It supports cheap and efficient computation of affine mapping operations and intersections with half-spaces. The system under consideration in this work represents a more complex system than Neural Network Control Systems (NNCS) previously considered in other works, and consists of a total of four components. Our experimental evaluation uses four different scenarios to show that our star-set based methods are scalable and can be efficiently used to analyze the safety of real-world cyber-physical systems (CPS).

III.1 Introduction

The last several years have seen a significant increase in the development of verification, testing, and falsification methods for systems that make use of neural networks. A comprehensive review of these techniques can be found in the survey by Liu et al. [60] and Xiang et al. [4]. While a great deal of these methods deal with networks in isolation, in recent years several methods have been proposed for verifying neural network control systems [50, 49, 52, 53, 54]. Neural network control systems commonly appear in safety critical systems, where the neural network controller is generated through the use of reinforcement learning and learning by demonstration [49]. In this realm, the safety verification problem is postulated as a reachability problem. Here the challenge is to estimate the set of reachable states of the closed loop system where the plant is modeled using linear ordinary differential equations and the controller is a feed-forward neural network. Despite significant progress in neural network control systems verification, developing a scalable methodology remains a key challenge.

This chapter examines the problem of verifying the safety of an unmanned underwater vehicle whose mission is to autonomously navigate without collisions. Our approach is based on the use of the star-set, which determines the exact output reachable set of the closed-loop system [108]. The utilization of the star-set in this work is largely due to the fact that the star-set is a computationally efficient set representation

adept at characterizing large input spaces, and supports cheap and efficient computation of affine mapping operations, and intersections with half-spaces [72]. Additionally, the system that we consider in this work represents a more complex system than *Neural Network Control Systems* (NNCS) previously considered in other works, consisting of a total of four components. Our experimental evaluation using four different scenarios demonstrates that our star-set based methods are scalable and can be efficiently used to analyze the safety of real-world *Cyber-Physical Systems* (CPS) for up to 30 second time windows.

In Section III.1.1, we discuss the motivation to perform this line of research as well as the specific challenge to verify. In Section III.2 we present the architecture considered as well as a detailed description of each individual component, Section III.3 presents the reachability algorithms used to verify NNCS and the difference with the one used for the UUV NNCS. Finally, Sections III.4 and III.5 presents the evaluation and verification results of the unmanned underwater vehicle.

III.1.1 Problem Formulation

Research in recent years has demonstrated that acquiring pipeline inspection data from *Unmanned Underwater Vehicles* (UUV) offers superior data quality and consistency, since UUV's often move in a highly efficient and stable manner. Thus, the unmanned methodology represents a paradigm shift in offshore geophysical survey and pipeline inspection. By utilizing a small UUV, the carbon footprint per inspection line kilometer can be reduced dramatically. A traditional host vessel may use 10,000 - 15,000 liters of fuel per day of operations, whereas a small unmanned craft uses up to 95% less fuel [109].

In this work, the problem we wish to consider is verifying that the UUV operates safely by successfully avoiding obstacles while executing its mission of inspecting a given pipeline. The obstacles we consider are static obstacles of varying sizes, and the safety specification we wish to consider is that the UUV does not move within a δ radius of a given obstacle. Mathematically, the safety specification, p, is satisfied if at every time step $t \in [t_0, t_f]$ in the bounded range $[t_0, t_f]$, governed by the start time t_0 and end time t_f,

$$\forall t \left(||X_v(t) - X_o(t)||_2 > \delta \right), \tag{III.1}$$

where X_v and X_o are the vehicle and obstacle positions, respectively.

Verification of the safety specification occurs at design time. Given an environment, an initial state and location for the UUV and an obstacle placed in front of it, our experiments evaluate the safety of the UUV.

III.2 System Architecture

The simulation experiments considered in this work were run using the *Robotics Operating System* (ROS) [110] and visualized using the Gazebo robot simulator [111]. The simulation package that we utilized is called

the *Unmanned Underwater Vehicle Simulator* (UUVS) [112], which provides a rich set of Gazebo plugins that are used to describe the dynamics and realistic simulation environments for surface and underwater vehicles. Additionally, a variety of Gazebo plugins were modified and added in an effort to more accurately capture hydrodynamic disturbance events and randomly place objects along the UUV path.

The UUV system in the Unmanned Underwater Vehicle Simulator, which we refer to as the *UUVS Model*, is made up of several components, shown in Figure III.1. There is a *Learning Enabled Component* (LEC), FNN Controller, which determines the control output for each time step. It utilizes two inputs, the distance to an object and the *Closest Point of Approach* (CPA), that are calculated based on observations from the forward-looking and side scan sonars. The controller then outputs 4 values, 2 regarding the heading change and 2 regarding the speed commands, which are then fed through a highly nonlinear normalization function, mapping these 4 values to a control command made up of a desired heading change (rad) and desired speed (m/s). These control commands are then processed by a low-level PID controller that converts these commands into actuation commands for the UUV.

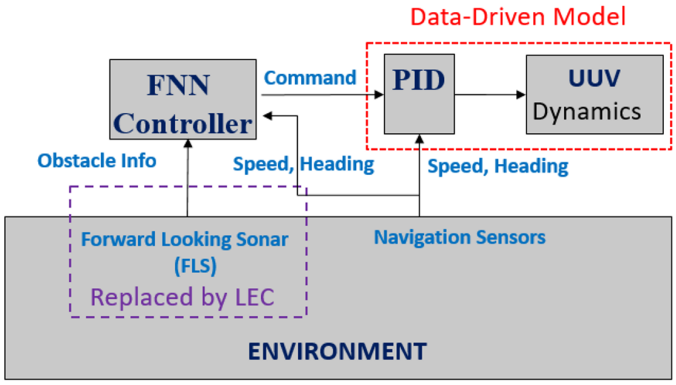


Figure III.1: UUV system architecture with the proposed modification for reachability analysis suitability with current available methods.

Combining the nonlinearities of the PID, the normalization function, and the sonar sensors, makes the task of formally verifying the *UUVS Model* highly complicated. These complexities make performing safety verification via reachability analysis intractable. Therefore, we created a modified model that approximates the performance of the *UUVS Model* and allows us to perform reachability analysis more conveniently. We will refer to this model as our *Verifiable Model*, and it is composed of the four parts shown in Figure III.4, *Feed Forward Neural Network Sensor*, *Feed Forward Neural Network Controller*, *Feed Forward Neural Network Normalization*, and a plant model.

III.2.1 Feed Forward Neural Network Sensor

We replace the Forward Looking Sonar (FLS) with a feed-forward neural network with 3 inputs, 7 hidden ReLU layers consisting of 10 neurons each, and a linear output layer with 2 neurons. This change corresponds to the purple dashed rectangle in Figure III.1. From the plant's outputs and the obstacle location, the neural network is able to compute both values needed for the controller's input, i.e., the distance to the obstacle and the CPA.

III.2.2 Feed Forward Neural Network Controller

The Feed Forward Neural Network Controller (FNN Controller) is the same in the *UUVS Model* and in the *Verifiable Model*. The FNN Controller is a Reinforcement Learning controller that was trained using the Vanilla Policy Gradient Method and optimized to maximize the reward function enforcing the certain conditions. If there is no object in the UUV's path, then the UUV is expected to maintain its current heading. Any deviation from this heading results in a penalty. However, if there is an obstacle within the UUV's path, then the UUV is penalized with respect to the closest point of approach and actions that change the UUV's heading are incentivized. Thus, the reward is associated with a given state, i.e., the input to the FNN Controller.

This neural network consists of 2 hidden layers of 32 fully connected, ReLU activated neurons. The two inputs are the distance to the obstacle and the closest point of approach. The linear output layer is made up of two values related to the heading change (rad) and two related to the speed (m/s).

III.2.3 Feed Forward Neural Network Normalization

The output of the FNN Controller is connected to this neural network in order to transform the 4 values provided into 2 outputs commands, the desired heading change (rad) and the desired speed (m/s). This feedforward neural network consists of 3 ReLU hidden layers with 50 neurons total and a linear output layer with 2 neurons. In Figure III.1, we add this neural network following the controller as a substitute of the highly nonlinear normalization function.

III.2.4 Plant Model

We replace the highly nonlinear UUV dynamical model with a simpler 2-dimensional linear discrete-time data-driven model obtained via system identification techniques [113]. We are able to consider a planar model (no z component) due to the constant depth that the UUVS is able to maintain due to the fixed constant depth command sent to the low-level PID controller. The data used in the system identification was obtained by performing a series of diverse maneuvers in the simulator and recording the data at fixed intervals. Because of

the way the data was collected, the model of the vehicle's dynamics also includes the low-level PID controller, which further makes the reachability analysis more amenable. The result is our plant model, which is defined by a set of discrete linear difference equations of the form

$$x(t+1) = Ax(t) + Bu(t) \tag{III.2}$$

$$y(t) = Cx(t) + Du(t), \tag{III.3}$$

where t is the time step, and A, B, C and D are constant matrices of size 8×8, 8×2, 3×8, and 3×2 respectively. The model is learned using classical system identification methods [113], and treat the dynamics to be learned as a *black-box* model. Initially, none of the system parameters are known [114].

We validate our plant model with respect to the results from following the same control inputs used by the Gazebo-simulated vehicle, choosing a different scenario from the previously collected trajectories. The recorded trajectory and the respective simulated one from our data-driven model are shown in Figure III.2.

The results show that, given the same inputs and initial state, our linear model captures the behavior of the Gazebo vehicle very well. For simplicity, we do not include any obstacles in the graph, only the (x,y) trajectories of the simulated vehicle and the identified vehicle. Normally, an average error of $\approx 0.6m$ would be considered large. However, the UUV modeled is $2m$ long and travels at an average of $1.5m/s$. Therefore, the error is relatively small compared to the size of the problem.

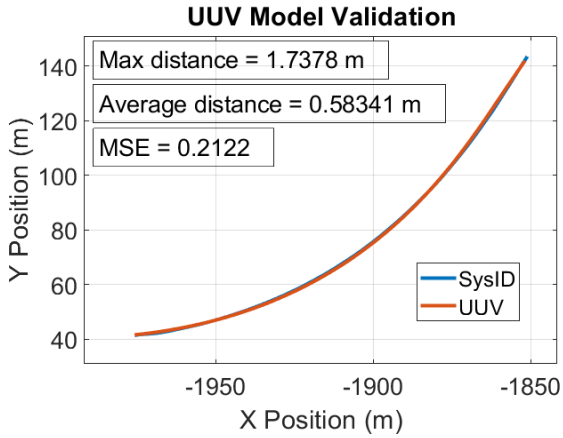


Figure III.2: Model validation: the orange line denotes the UUV trajectory recorded by the ROS simulation. The blue trajectory denotes the trajectory of the data-driven model obtained via system identification. This is the model we use in our MATLAB experiments. We include the average and maximum distance between each x-y position of the ROS trajectory and the MATLAB simulation at each point in time. We also show the mean-square error (MSE) between both trajectories.

III.3 Reachability Analysis of Neural Network Control Systems

A standard architecture of a reachability analysis problem for a *Neural Network Control System* (NNCS) is displayed in Figure III.3 and is formulated as follows. Starting from an initial set of states X_0 for our model P, the controller C takes the output set of the plant Y_p as an input to compute the controller output set $U = F(Y_p)$. Then, the control set (controller output set) is applied to the plant to compute the next set of states $X_{k+1} = AX_k + BU_k$, where $k \in [0,t]$.

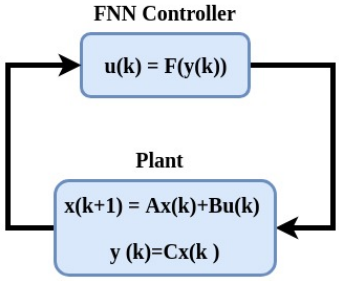


Figure III.3: Neural network control system (NNCS).

This process is then repeated iteratively to obtain a sequence of reachable sets of states, $X_0, X_1, X_2, ..., X_t$, where X_0 is the initial state and X_t are the reachable states at time t. To obtain precise reachable sets for the NNCS, we compute the exact control set U given the output set Y_p. Also, we compute the exact set of reachable states X_i given the previous set of states X_{i-1} and the corresponding control set U_{i-1}. To compute these exact reachable sets, we utilize the star-set set representation to describe them, since the star-set is computationally less expensive than other common set representations such as polytopes and zonotopes [108]. We refer the reader to the following papers for an in-depth discussion of star-set based techniques [108, 55].

At each time step, given the states of the learned model of the UUV, the location of the obstacle, the distance to the obstacle, the closest point of approach to the obstacle with respect to the UUV trajectory, we pass these values to the FNN Controller, and compute its output set. It is worth noting that if an obstacle is observed by the FLS then the distance to the obstacle is returned, otherwise a constant value is returned for the closest point of approach and the distance to the object. The controller output set is fed through the normalization component, the other NN of the system, which computes an output set in terms of heading change and speed. This is then fed through the model dynamics, which computes the reach sets for the outputs. From these, we only take the x and y position intervals, as these are the only ones needed to calculate the data for the feedback loop back to the controller.

III.3.1 Reachability algorithm for NNCS

We are able to compute the exact reachable set of the NNCS by computing the exact control set and the exact plant output set at each time step. However, in obtaining the reachable sets for a complex system, the number of sets increase quickly over time, making it computationally more and more expensive to obtain reachable sets for successive time steps. Therefore, this process is very time-consuming even through the use of time optimizations such as parallel computing.

To avoid the explosion of the number of sets needed to represent the reachable set, we take a single convex hull of the reachable sets of states of the plant after each step, and then we compute the output reachable set of the plant $(Y = CX)$, which is then fed back to the next component as a single star set. However, the tradeoff here is that the solution to this problem is no longer exact.

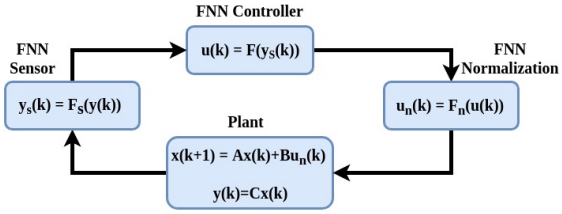


Figure III.4: Modified neural network control system for the analysis of the unmanned underwater vehicle. We compute the exact output reach set of all 3 FNN, but over-approximate the plant's output.

III.3.2 Reachability algorithm for UUV System

Our *Verifiable Model* uses three LEC's and a singular plant model connected as shown in Figure III.4. Due to the increased number of LEC's in our system, we modified the general algorithm for NNCS to add all the components in the system according to Algorithm 1.

We have four main steps in the reachability analysis computation. (1) We compute the output set of the plant given the initial state. (2) Using the location of the UUV output from the plant, we compute the exact output set of the *sensor* neural network, FNN Sensor. (3) We compute the exact output set of the controller, divided into 2 operations, the main controller, FNN Controller, and the normalizing neural network, FNN Normalization. (4) We compute the set of approximate states of the plant at the next time step.

III.4 Evaluation: Collision Avoidance

For our underwater vehicle system, we present several scenarios in which we modify the initial states and the location of the obstacle to be avoided. The obstacle is located in front of the vehicle at the beginning of the experiment, and the objective is for the vehicle to detect it and avoid a collision with it.

Based on the assumption that our vehicle maintains a constant depth, we only consider the x and y

Algorithm 1: Reachability Algorithm of the UUV Control System.

```
% k_max: Number of steps
% A,B,C: plant's matrices
% I: Set of initial states for the plant
% X: Set of current states for the plant
% Y: Output set of the plant
% R: Output reachable set of the plant
% F: Neural network controller
% N: Neural network normalizing output controller
% S: Neural network as sensor function
% D: Set of unsafe states

INITIALIZE
R = cell(1,k_max +1);
R{1,1} = I;
X_1 = R{1,k};

REACH COMPUTATION
for k = 1 : k_max do
    Y_k = CX_k;
    S_out = S(Y_k);
    F_out = F(S_out);
    N_out = N(F_out);
    for j = 1:length(N_out) do
        X_{k+1}^j = AX_k + BN_out^j;
    X_{k+1} = IntervalHull(X_{k+1});
    R{1,k+1} = X_{k+1};
    if (X_{k+1} ∩ D) ≠ ∅ then
        return;
```

positions of the obstacles. The obstacles placed in the UUV path are boxes with 1 meter long edges. The safety specification p, formally defined in Equation III.1, that we analyze is that the vehicle will never travel within a $\delta = 2$ meter radius of the center of the object.

Given an initial state set and an obstacle location, we calculate the output reachable sets for a k_{max} steps until the vehicle violates the safety specification or until the vehicle has cleared the obstacle and the safety specification has not been violated.

III.5 Results

Here, we experiment with four different scenarios where the vehicle has been placed at a different location with different initial states, as well as modifying the location of the obstacle [1]. These four scenarios generally represent a wide range of the scenarios the UUV might encounter. Since the FNN Controller makes decisions based solely on the distance between the UUV and the obstacle and the CPA, these scenarios can be generalized to any case where the UUV is at the same distance with the same CPA.

In all four scenarios, we pick an initial point, (x, y, yaw), from a trajectory generated in UUVS. From this starting point, we estimate the initial states of the plant using MATLAB's System Identification Toolbox [114]. We take these 8 initial states (estimated) and set them as the initial point for the UUV trajectory in UUVS, which is marked in blue. Additionally, the starting point of the UUV in UUVS is used to generate the no object path ($N.O.P.$), which is the magenta dashed line representing the projected path the UUV would travel if it were to keep its orientation, (yaw), constant. Displaying the N.O.P. helps visualize how the FNN Controller directs the UUV away from the obstacle. For the reachability analysis, we take the estimated initial states and create lower and upper bounds around these points by adding -0.0001 and +0.0001 to each state, respectively. The blue boxes in the result figures visualize the reachable states at each time step.

III.5.1 Experiment 1: A Minor Course Correction

In Experiment 1, shown in Figure III.5, the UUV starts with a trajectory that does not intersect with the obstacle. Therefore, little to no course correction is needed to avoid the obstacle.

We observe this behavior in Figure III.5 as the UUV meets the specification and clears the obstacle after 24 seconds.

The exact initial state (x_0) intervals and the location of the obstacles (P_{obs}) are the following:

[1] Code at https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/WAAS2020

$$x_0 = \begin{bmatrix} \begin{bmatrix} -0.044174880802800 & , & -0.043974880802800 \end{bmatrix} \\ \begin{bmatrix} 0.007009187707955 & , & 0.007209187707955 \end{bmatrix} \\ \begin{bmatrix} 0.003998288293794 & , & 0.004198288293794 \end{bmatrix} \\ \begin{bmatrix} -0.066590984864864 & , & -0.066390984864864 \end{bmatrix} \\ \begin{bmatrix} 0.076075384291567 & , & 0.076275384291567 \end{bmatrix} \\ \begin{bmatrix} -0.063045980165905 & , & -0.062845980165905 \end{bmatrix} \\ \begin{bmatrix} -7.781701966436715e-04 & , & -5.781701966436714e-04 \end{bmatrix} \\ \begin{bmatrix} 0.095613905722162 & , & 0.095813905722162 \end{bmatrix} \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -1888.260 \\ 89.443 \end{bmatrix}$$

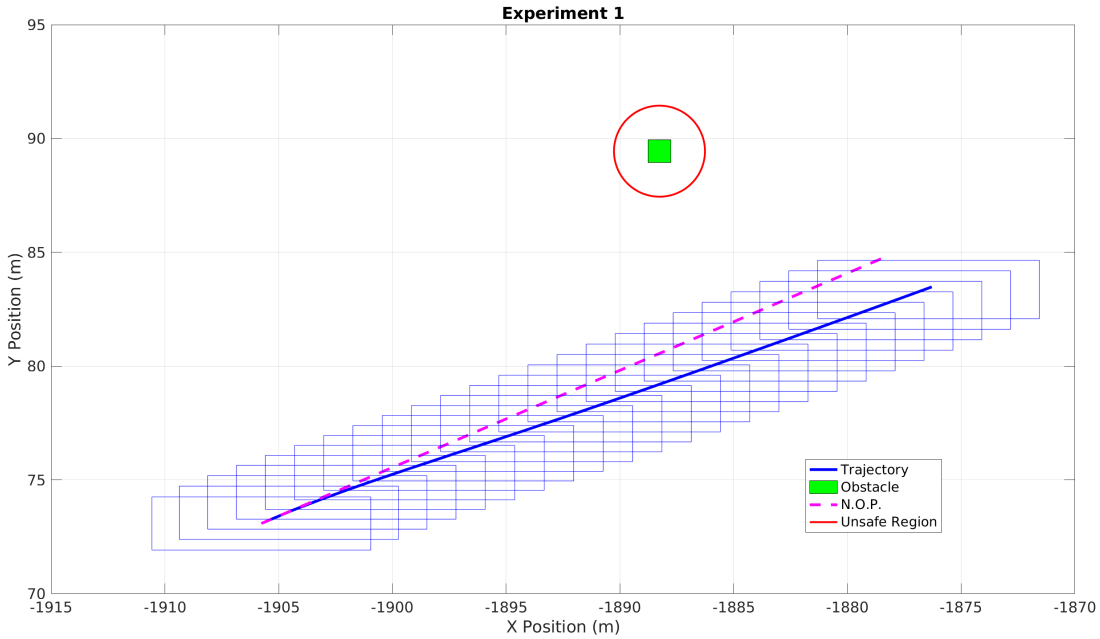


Figure III.5: The UUV is initialized at *(-1905.577, 73.085)* pointing *23.115* degrees from horizontal. The obstacle is generated at *(-1888.260, 89.443)*. The actual trajectory of the UUV stays within the computed reachable sets, and the UUV passes the obstacle after 24 seconds.

III.5.2 Experiment 2: Immediate Course Correction

In Experiment 2, we initialize the UUV with a path directed towards the middle of the obstacle. In order to avoid a collision, the UUV must make an immediate course correction. The UUV is able to detect the object

and as a result, it manages to turn and deviates its trajectory to avoid the obstacle in a safe manner as shown in Figure III.6.

The exact initial state (x_0) intervals and the location of the obstacles (P_{obs}) are the following:

$$x_0 = \begin{bmatrix} \begin{bmatrix} -0.039780351857121 & , & -0.039580351857121 \end{bmatrix} \\ \begin{bmatrix} 0.062835250855941 & , & 0.063035250855941 \end{bmatrix} \\ \begin{bmatrix} 0.151063041362560 & , & 0.151263041362560 \end{bmatrix} \\ \begin{bmatrix} 0.829311305915308 & , & 0.829511305915308 \end{bmatrix} \\ \begin{bmatrix} -0.628460030147656 & , & -0.628260030147656 \end{bmatrix} \\ \begin{bmatrix} 0.283328195630901 & , & 0.283528195630901 \end{bmatrix} \\ \begin{bmatrix} -0.042414499938101 & , & -0.042214499938101 \end{bmatrix} \\ \begin{bmatrix} -0.304932444834164 & , & -0.304732444834164 \end{bmatrix} \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -2099.480 \\ -431.911 \end{bmatrix}$$

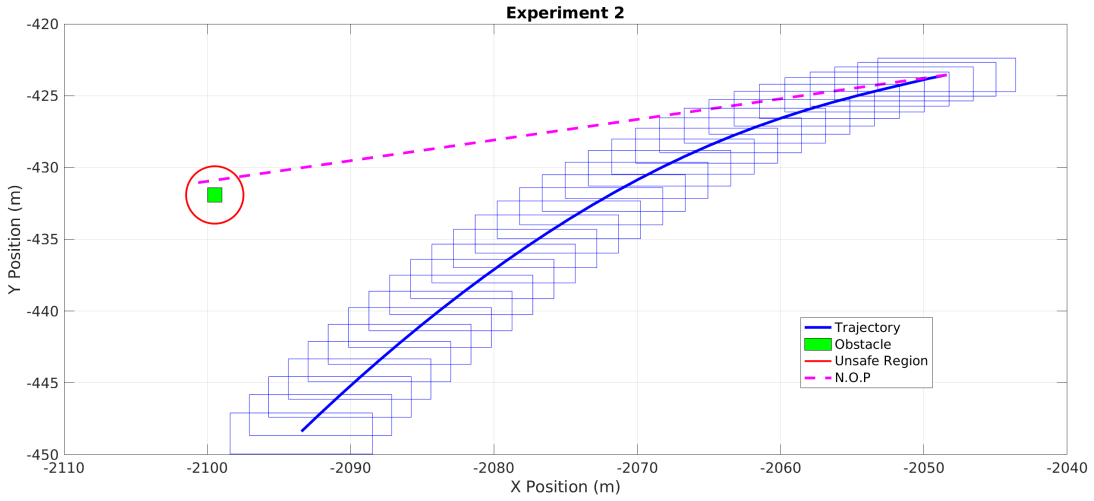


Figure III.6: The UUV is initialized at *(-2051.187, -423.978)* pointing *-170.256* degrees from horizontal. The obstacle is generated at *(-2099.480, -431.911)*. The actual trajectory of the UUV stays within the computed reachable sets, and the UUV passes the obstacle after 30 seconds.

III.5.3 Experiment 3: Unsafe Scenario

In Experiment 3, we initialize the UUV within 17m of the obstacle and heading towards it. This represents a scenario where the UUV does not satisfy the safety condition. We can see that at the eleventh step, 11

seconds, the intersection of the vehicle's reachable set and the unsafe set is not null. However, based on the over-approximation method used in the reachability analysis, these results are inconclusive.

The exact initial state (x_0) intervals and the location of the obstacles (P_{obs}) are the following:

$$x_0 = \begin{bmatrix} \begin{bmatrix} -0.039780351857121 & , & -0.039580351857121 \end{bmatrix} \\ \begin{bmatrix} 0.062835250855941 & , & 0.063035250855941 \end{bmatrix} \\ \begin{bmatrix} 0.151063041362560 & , & 0.151263041362560 \end{bmatrix} \\ \begin{bmatrix} 0.829311305915308 & , & 0.829511305915308 \end{bmatrix} \\ \begin{bmatrix} -0.628460030147656 & , & -0.628260030147656 \end{bmatrix} \\ \begin{bmatrix} 0.283328195630901 & , & 0.283528195630901 \end{bmatrix} \\ \begin{bmatrix} -0.042414499938101 & , & -0.042214499938101 \end{bmatrix} \\ \begin{bmatrix} -0.304932444834164 & , & -0.304732444834164 \end{bmatrix} \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -2099.480 \\ -431.911 \end{bmatrix}$$

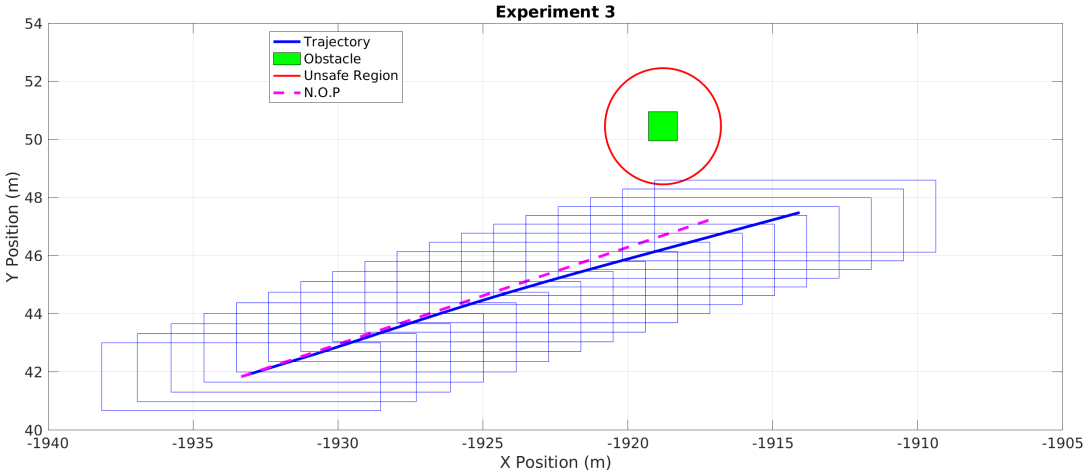


Figure III.7: The UUV is initialized at *(-1933.337, 41.836)* pointing *18.478* degrees from horizontal. The obstacle is generated at *(-1918.790, 50.456)*. The actual trajectory of the UUV stays within the computed reachable sets, and the UUV passes the obstacle after 11 seconds.

III.5.4 Experiment 4: Choosing a New Path

In Experiment 4, the UUV is initialized far enough away from the obstacle that a small change in the initial heading finds a clear and open path that does not require further adjustments. This highlights the learned

behavior of the FNN Controller to minimize heading changes while maximizing the distance between the UUV and the obstacle. Behavior like this could be considered undesirable and is the reason why designing a reward function is so important to the performance of a reinforcement learning controller[2].

However, we clearly observe that the UUV avoids the obstacle and does not get closer than 12 meters to the center of the obstacle.

The exact initial state (x_0) intervals and the location of the obstacles (P_{obs}) are the following:

$$x_0 = \begin{bmatrix} \begin{bmatrix} -0.044991857837650 & , & -0.044791857837650 \end{bmatrix} \\ \begin{bmatrix} 0.010793055476382 & , & 0.010993055476382 \end{bmatrix} \\ \begin{bmatrix} 0.017457598529114 & , & 0.017657598529114 \end{bmatrix} \\ \begin{bmatrix} 0.022151317193675 & , & 0.022351317193675 \end{bmatrix} \\ \begin{bmatrix} 0.101951191993133 & , & 0.102151191993133 \end{bmatrix} \\ \begin{bmatrix} -0.280253273679023 & , & -0.280053273679023 \end{bmatrix} \\ \begin{bmatrix} 0.172975572623461 & , & 0.173175572623461 \end{bmatrix} \\ \begin{bmatrix} 0.195338830333452 & , & 0.195538830333452 \end{bmatrix} \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -1918.790 \\ 50.456 \end{bmatrix}$$

III.6 Summary

This chapter presents an efficient over-approximate reachability scheme that consists of an exact method for the reachability analysis of neural networks, and an over-approximate method used in the plant reachability analysis in order to consider the safety verification of a cyber-physical system with an RL controller. The safety specification is defined based on the location of obstacles and the underwater vehicle. We have shown that our method is scalable for real-world applications in our analysis of the obstacle avoidance capabilities of an unmanned underwater vehicle.

[2]For more information about this problem, look into work discussing *Reward Shaping*.

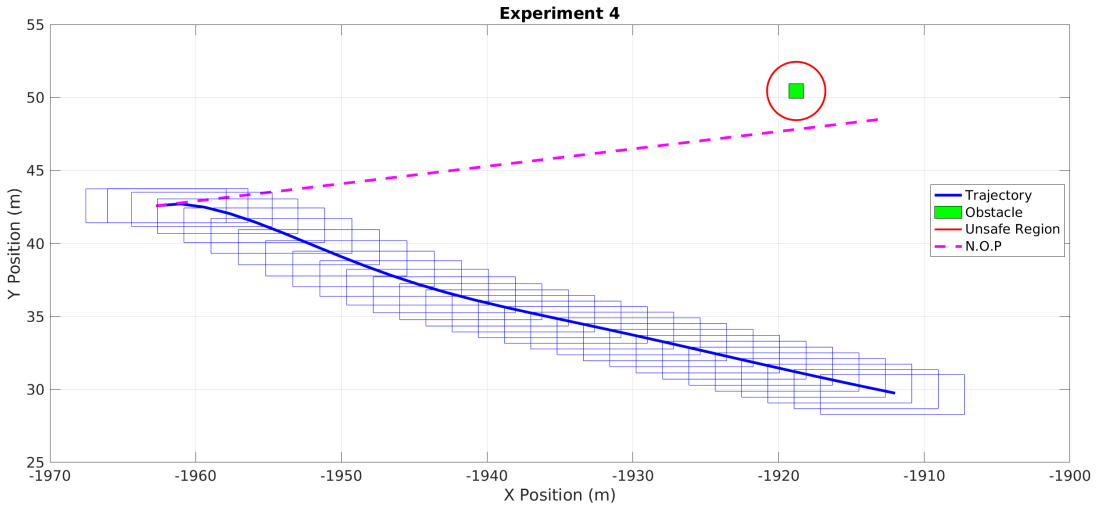


Figure III.8: The UUV is initialized at *(-1948.779, 43.165)* pointing *6.807* degrees from horizontal. The obstacle is generated at *(-1918.790, 50.456)*. The actual trajectory of the UUV stays within the computed reachable sets, and the UUV passes the obstacle after 29 seconds.

CHAPTER IV

Safety Verification of Air to Air Unmanned Collision Avoidance System

This chapter is adapted from the material presented in [16].

In this chapter, we analyze a NNCS with switching classification-based controllers synthesized from a set of lookup tables that advise advisory commands to unmanned aircraft. An example of such systems is the unmanned Airbone Collision Avoidance System (ACAS Xu), which is a very popular benchmark for open-loop neural network control system verification tools. We propose a new closed loop extension of this benchmark, which consists of a set of ten closed loop properties selected to evaluate the safety of an ownship aircraft in the presence of a co-altitude intruder aircraft. These closed loop safety properties are used to evaluate 5 of the 45 neural networks that comprise the ACAS Xu benchmark (corresponding to co-altitude cases) as well as the switching logic between the 5 neural networks. The combination of nonlinear dynamics and switching between five neural networks is a challenging verification task accomplished with star set reachability methods in two verification tools: NNV and nnenum. The safety of the ownship aircraft under initial position uncertainty is guaranteed in every scenario proposed.

IV.1 Introduction

The use of machine learning in information technology domains has drastically improved automated capabilities like natural language processing [115], image classification [116], and object detection [117]. In the last several years, machine learning has also tackled complex game playing with high dimensional state spaces, beating world experts in Go [118, 119] and StarCraft II [120]. In addition to potentially optimizing performance of complex systems, neural networks are often much smaller and faster to compute than other optimization algorithms, making them attractive for use on Cyber-Physical Systems (CPS) like robots, cars, drones, and satellites. For example, neural networks have been used to compress the 2 gigabytes of lookup tables used by the Airborne Collision Avoidance System X (ACAS X) to 5 megabytes of neural network weights [121]. However, verification of machine learning in safety-critical system domains has historically been limited to frameworks that treat neural networks like black boxes. Neural networks (NN) are not black boxes, they compute deterministic mathematical functions that are increasingly amenable to formal reasoning methods [60, 4]. While many of these methods focus on analyzing the networks in isolation, several recent methods attempt to verify neural network control systems (NNCS)

[50, 49, 52, 53, 54, 58, 56, 122, 55, 14, 15, 81, 82].

Despite the recent progress in NNCS verification, developing scalable and non-conservative methods remains a key issue. Due to this growing trend and remaining challenges, several friendly competitions have been recently organized to compare the performance of these verification tools, including [123] for NN verification and [124] for NNCS verification. In addition to these competitions, studies by Ivanov et al. [78], Tran et al. [55] and Manzanas Lopez et al. [15], have investigated the limits and capabilities of existing verification methods and tools [50, 14], and the trade-offs between scalability and conservativeness. Similar to the proposed NNCS in this work, Julian and Kochenderfer [75] and Akintunde et al. [56, 122] have formally verified another collision avoidance system for aircraft. Specifically, a closed-loop variant of the simplified version of the vertical avoidance control system in aircraft, the Vertical Collision Avoidance System (VerticalCAS) [80], is verified using reachability analysis and Mixed Integer Linear Programming (MILP) methods respectively. In addition to the VerticalCAS, Julian and Kochenderfer [125, 79] presented the same formal approach to verify the safety of Horizontal Collision Avoidance System (HorizontalCAS), which is another advisory system inspired by early prototypes of the ACAS Xu and presents the same input-output space of the ACAS Xu system, but with smaller size neural networks. Irfan et al. [81] formally verified the advisory system for small unmanned aircraft [126] (ACAS sXu) via symbolic interval reachability analysis with ReluVal [83]. The research studied a different state space and was composed of smaller neural network controllers than the ACAS Xu NNCS presented here. Similarly, Claviere et al. [82] use ReluVal to formally analyze one test case of the ACAS Xu NNCS in combination with a validated simulation approach to approximate the reachable sets of the plant.

This manuscript aims to extend and improve the verification results of these studies to a comprehensive set of benchmarks in which two aircraft are travelling at the same altitude. These benchmarks are evaluated using star set reachability methods via the NNV [14] and nnenum [127] NNCS verification tools, which improve the scalability and over-approximation tightness of the symbolic interval analysis approaches. To the best of our knowledge, without any additional modifications or enhancements to existing NNCS reachability tools, these are the only NNCS tools that are able to verify this benchmark due to the switching behavior of the classification-based controllers. These improvements are specially notable in 7 of the 10 test cases where the intruder is approaching the ownship from either side. The symbolic interval analysis methods in [82] are only able to verify 75% of the area considered in these cases, while increasing the verification time between 1 and 2 orders of magnitude with respect to other areas. On the other hand, the star set methods utilized in this work are able to verify the safety of the entire region considered for each of these test cases whilst maintaining a similar computation time as the other experiments.

The main contributions are:

- A description of the ACAS Xu neural network compression closed loop benchmark and definition of ten closed loop verification properties.
- Improved scalability and over-approximation tightness in star-set reachability methods by developing a verification approach for neural network closed loop systems with switching classification-based controllers using star sets.
- Successful verification of 5 of 45 ACAS Xu Neural Networks corresponding to cases with no vertical separation, and the switching behavior against all ten safety properties using two reachability analysis tools: NNV and nnenum.

The rest of the chapter is organized as follows. Section IV.2 describes the ACAS Xu NN compression system, and Section IV.3 describe the plant model used as well as the NNCS benchmark. Section IV.4 describes the closed loop test case generation of safety properties, while in Section IV.5 the evaluation approaches used are defined. Finally, section IV.6 introduces the results and section IV.8 provides the summary of the contributions of this chapter.

IV.2 ACAS X

The Airborne Collision Avoidance System X (ACAS X) is under development to one day replace the Traffic Collision Advisory System II (TCAS II) as a mid-air collision prevention system [128]. ACAS X is designed to be compatible with the FAA's Next-Generation Air Transportation System (NextGen), which uses new sensing and navigation technologies coupled with new procedures to better optimize air traffic [128].

There are five variants of ACAS X [128], and this manuscript provides a collection of 10 closed loop properties to analyze the safety of a neural network approximation of the ACAS Xu variant:

- ACAS Xa (active): Designed to provide protection from all tracked aircraft using onboard sensors on large manned aircraft, ACAS Xa issues alerts and vertical advisories to the pilot [129].
- ACAS Xu (unmanned): Optimized for unmanned aircraft systems (UAS), ACAS Xu issues turn rate advisories to remote pilots [130].
- ACAS Xo (operation): special alerts during operations such as parallel runway approaches [128].
- ACAS Xp (passive): tracks aircraft for potential collisions, but doesn't produce advisories [128].
- ACAS sXu (small unmanned): Designed for small UAS (sUAS) to avoid collision with manned aircraft, UAS, and other sUAS [126].

Both ACAS Xa and ACAS Xu have a goal to avoid near midair collisions (NMAC) [131], defined as separation less than 100 ft vertically and 500 ft horizontally [132], as depicted in Fig. IV.1.

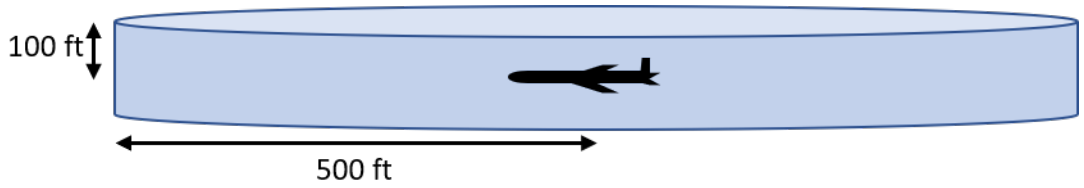


Figure IV.1: Near midair collision cylinder, defined as separation less than 100 ft vertically and 500 ft horizontally.

ACAS X functionality centers on the use of a set of look up tables generated offline with dynamic programming and Markov decision processes (MDPs) [133]. The input to these lookup tables is a probabilistic state distribution of the relative position and velocity of nearby aircraft, approximated from sensor data. The output of the lookup tables is a cost used by ACAS X to decide the optimal advisory to provide to the pilot: no alert, traffic advisory, or a resolution advisory for pilots to maintain or increase safe separation from other aircraft [128].

This manuscript focuses on analysis of the ACAS Xu variant of ACAS X, described briefly in Section IV.2.1, and specifically, a neural network compression of this ACAS Xu algorithm, described in Section IV.2.2. The ACAS Xu neural networks receive actual states of the system, rather than probabilistic state distributions from sensor data, and output the most optimal action.

IV.2.1 ACAS Xu

ACAS Xu, the unmanned aircraft version, assigns values to a set of output actions based on a set of input variables as described in Table IV.1. The first five variables describe 2D considerations, the sixth variable brings the scenario into 3D, and the seventh variable promotes advisory selection consistency. In the initial lookup tables, the state variables are discretized in a seven dimensional grid with 120 million points [134] that assign a value to each of 5 different output action options, resulting in 600 million floating point numbers. When the state of the system falls between discrete points, the nearest neighbor point is used [134]. To deal with sensor uncertainty in the system state, ACAS Xu uses an unscented Kalman filter [135].

IV.2.2 Neural Network Compression of ACAS Xu

A major challenge to the implementation of ACAS Xu is that the initial lookup tables require hundreds of gigabytes of floating point storage [121]. Using a technique called downsampling, values may be removed from the table in areas where variation between the values is very small and has been shown to reduce ACAS

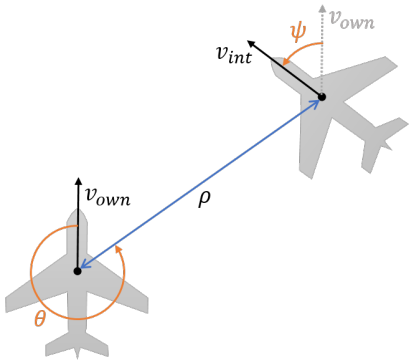


Figure IV.2: Diagram of the ACAS Xu physical variables.

Table IV.1: Input state variables used in ACAS Xu.

Variable	Units	Description	Tables	NN
ρ	ft	distance between ownship and intruder	Y	Y
θ	rad	angle to intruder w.r.t ownship heading	Y	Y
ψ	rad	heading of intruder w.r.t. ownship heading	Y	Y
v_{own}	ft/s	velocity of ownship	Y	Y
v_{int}	ft/s	velocity of intruder	Y	Y
τ	s	time until loss of vertical separation	Y	N
a_{prev}	deg/s	previous advisory	Y	N

Table IV.2: ACAS Xu Actions (Horizontal Collision Avoidance).

Action	Description
SL	strong left at 3.0 deg/s
WL	weak left at 1.5 deg/s
COC	clear of conflict (do nothing)
WR	weak right turn at 1.5 deg/s
SR	strong right turn at 3.0 deg/s

Xu table size to approximately 2 gigabytes [121]. For ACAS Xa, which limits advisories to vertical maneuvers and has smaller lookup tables to begin with, downsampling was sufficient [136]. However, ACAS Xu's 2 gigabyte size may still be too large for the storage constraints of certified avionics hardware on UAS[125].

Developed in [121] and evaluated in [130], 45 separate neural networks were used to compress the lookup table. Each network is denoted $N_{\gamma,\beta}$, where γ corresponds to the index (1 to 5) of a specific value of previous advisory $a_{prev} \in \{COC, WL, WR, SL, SR\}$ and β corresponds to the index (1 to 9) of a specific value of time to loss of vertical separation $\tau \in \{0, 1, 5, 10, 20, 40, 60, 80, 100\}$ seconds. For example, $N_{2,3}$ corresponds to a neural network in which $a_{prev} = WL$ and $\tau = 5$. Then each of these networks receives inputs for the remaining five state variables (ρ, θ, ψ, v_{own}, and v_{int}) and outputs a value associated with each of the five output variables ($\{COC, WL, WR, SL, SR\}$). Several architectures and optimizers were considered and analyzed for the training

of all the 45 neural networks. AdaMax, a variant of Adam, was chosen as it proved to learn the fastest without getting stuck in local optima. As for the layer architecture, six hidden layers of 50 neurons each proved to yield the best results while maintaining an efficient computation time [121]. Hence, all the neural networks have five inputs, five outputs and six hidden layers of 50 neurons each, as depicted in Fig. IV.3.

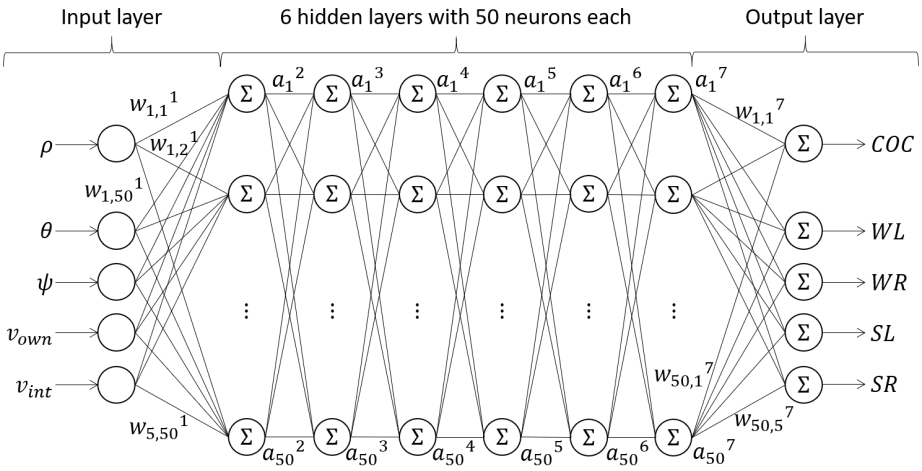


Figure IV.3: Depiction of the ACAS Xu neural network.

IV.2.3 Open-Loop vs. Closed-Loop Verification

Previous research has verified the neural networks using a set of open-loop test points [130, 134, 80, 125]. In other words, for a single instant in time, the neural network is verified to output appropriate scores for a particular set of inputs. The open-loop properties described in [130] became a popular benchmark for open loop verification, and are summarized here in Section IV.2.5. However, a much more important property that needs to be verified for these neural networks is that, when used in *closed-loop* control, the neural network approximation of the lookup tables will not result in control actions that violate the NMAC safety constraint depicted earlier in Fig. IV.1. The development and description of a set of closed-loop safety properties is one of the main contributions of this chapter.

IV.2.4 Set-based Verification Compared to Monte Carlo and Design of Experiments

Set-based verification is a possible alternative to Monte Carlo simulation, which uses thousands or millions of random samples to evaluate performance across the system state space [137]. An alternative approach to Monte Carlo, especially when evaluating individual points is computationally expensive, is to apply Design of Experiments to sample the state space in a structured way rather than randomly [138]. Latin Hypercube can be applied to Monte Carlo or Design of Experiments approach to use a reduced sample spread evenly across the state space [139]. However, Monte Carlo and Design of Experiments are not as comprehensive as

set-based neural network verification, which is the focus of this manuscript.

IV.2.5 Open-Loop Properties

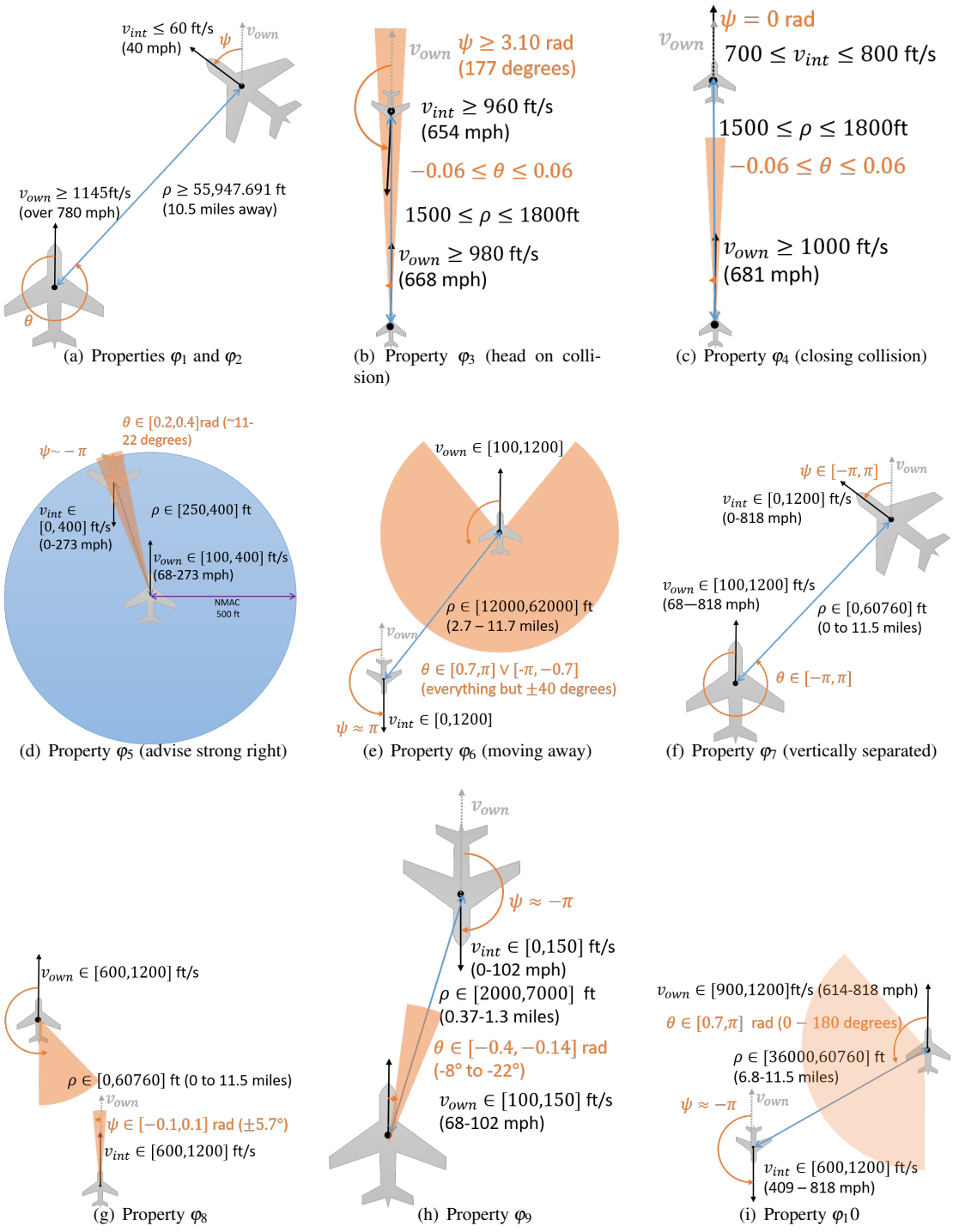
An ACAS Xu verification benchmark proposed 10 open loop properties of the neural network approximation. These properties are summarized with visual depictions of the geometry to give provide the reader with more intuition into how these neural networks have been verified in the past and how this compares to the closed loop test cases. It is worth noting that these neural networks assign a value to each of the five possible outputs, with the convention that the *lowest* output is the best choice (in other neural network designs, the highest input may be the best choice).

Table IV.3: ACAS Xu Benchmark Open Loop Properties.

	ρ (ft)	θ (rad)	ψ (rad)	v_{own} (ft/s)	v_{int} (ft/s)	Networks	Output
φ_1	$\geq$ 55,948	$\times$	$\times$	$\geq$1145	$\leq$ 60	All	COC $\leq$ 1500
φ_2	$\geq$ 55,948	$\times$	$\times$	$\geq$1145	$\leq$ 60	$N_{x\geq 2,y}$	COC not max
φ_3	$\in$ [1500,1800]	$\in [-0.06,0.06]$	$\geq$3.10	$\geq$980	$\geq$960	$\neq N_{1,y\geq 7}$	COC not min
φ_4	$\in$ [1500,1800]	$\in [-0.06,0.06]$	0	$\geq$1000	$\in [700,800]$	$\neq N_{1,y\geq 7}$	COC not min
φ_5	$\in$ [250,500]	$\in [0.2,0.4]$	$\approx -\pi$	$\in [100,400]$	$\in [0,400]$	$N_{1,1}$	SR min
φ_6	$\in$ [12000, 62000]	$\in [0.7,\pi] \vee \in [-pi,-0.7]$	$\approx \pi$	$\in [100,1200]$	$\in [0,1200]$	$N_{1,1}$	COC min
φ_7	$\in [0,60760]$	$\in [-\pi,\pi]$	$\in [-\pi,\pi]$	$\in [100,1200]$	$\in [0,1200]$	$N_{1,9}$	SL,SR min
φ_8	$\in [0,60760]$	$\in [-\pi,-0.75\pi]$	$\in [-0.1,0.1]$	$\in [600,1200]$	$\in [600,1200]$	$N_{2,9}$	WL $\vee$ COC
φ_9	$\in [2000,7000]$	$\in [-0.4,-0.14]$	$\approx -\pi$	$\in [100,150]$	$\in [0,140]$	$N_{3,3}$	SR min
φ_{10}	$\in [36000,60760]$	$\in [0.7,\pi]$	$\approx -\pi$	$\in [900,1200]$	$\in [600,1200]$	$N_{4,5}$	COC min

Property φ_1 may select a value of ρ (approximately 10.5 miles away) that is larger than the maximum turning radius of an aircraft going at 1145 ft/s (43,736). This maximum velocity of 1145 ft/s is approximately 780 miles per hour (mph), which as a conservative upper bound is much larger than what a commercial aircraft would typically fly (Mach 1 is 761 mph at sea level, decreasing to 678 mph at a 30,000 ft typical commercial aircraft cruise altitude). The 60 ft/s (40 mph) velocity of the intruder in this property is a conservative lowest velocity for a fixed wing aircraft. While the selection of variables seems reasonable, this first property isn't necessarily a good property because it specifies a value of the output which is an artefact of the design choices for the neural network rather than an ordinal ranking of the desired outcome for the aircraft encounter. In other words, a better property might specify that the selected action produced by the neural network should be clear of conflict. Property φ_2 is a slightly better variation of φ_1 because it tests that clear of conflict is not the last choice of the neural network for a subset of the conditions of φ_1.

Properties φ_3 and φ_4 checks that the neural network does not output clear of conflict in cases where the intruder is ahead of the ownship and a collision is imminent. Property φ_3 checks when the two aircraft are approaching for a head on collision within less than a second (unless there is sufficient vertical separation). Property φ_4 checks when the ownship is directly behind and closing on the intruder at a rate of 200-300 ft/s,



(a) Properties φ_1 and φ_2

(b) Property φ_3 (head on collision)

(c) Property φ_4 (closing collision)

(d) Property φ_5 (advise strong right)

(e) Property φ_6 (moving away)

(f) Property φ_7 (vertically separated)

(g) Property φ_8

(h) Property φ_9

(i) Property φ_10

Figure IV.4: Depiction of the aircraft encounter geometry for the open loop ACAS Xu verification properties

which will result in a collision in less than 9 seconds.

The remaining 6 properties check a wide range of conditions. Property φ_5 checks that the neural network favors the strong right advisory if the intruder is passing very close to the ownship's left. Property φ_6 checks that the neural network outputs clear of conflict if the intruder is sufficiently far away behind and moving away from the ownship. Property φ_7 check that the neural network does not output a strong left or right advisory when vertical separation is large ($\tau = 100$s) and the previous advisory was clear of conflict. Property φ_8 checks that the neural network outputs weak left or clear of conflict as the minimal score when the previous advisory was weak left, there are 100 seconds until a loss of vertical separation, and the intruder is behind and to the right of the ownship. Property φ_9 checks that the neural network outputs a strong left signal in a potential head on collision where the intruder is passing to the right. Property φ_{10} checks that the neural network outputs clear of conflict if the intruder aircraft is far away and moving away.

IV.3 Model

The verification effort in this chapter focuses on verifying 5 of 45 total neural networks corresponding to cases where the loss of vertical separation τ is 0 (N_{11}, N_{12}, N_{13}, N_{14}, and N_{15}). This reduces the problem to a two-dimensional co-altitude case. Nonetheless, verification of these neural networks and the switching between them present a significant challenge. Focusing on the co-altitude cases, the nonlinear plant model of the intruder and ownship aircraft are both represented using Dubins aircraft model described in Eq. IV.1.

$$\begin{aligned} \dot{x}_1 &= \dot{x} = v\cos(\psi), \\ \dot{x}_2 &= \dot{y} = v\sin(\psi), \\ \dot{x}_3 &= \dot{\psi} = u, \end{aligned} \tag{IV.1}$$

This is a simplified yet reasonable model of the aircraft dynamics at a similar level of abstraction as the inputs to the ACAS Xu NNCS, which represents the aircraft in terms of velocities, headings, and distances. However, the ACAS Xu NN model requires inputs in terms of distance ρ, angle to intruder from ownship θ, and heading of intruder with respect to ownship ψ. Conversion between the states of the Dubins model of each aircraft and the variables used as inputs to the NN is achieved via a set of complex nonlinear functions defined in Eq. IV.2.

$$
\begin{aligned}
\rho &= \sqrt{(x_{int} - x_{own})^2 + (y_{int} - y_{own})^2} \\
\theta &= 2tan^{-1}(\frac{y_{int} - y_{own}}{x_{int} - x_{own} + \rho}) - \psi_{own} \\
\psi &= \psi_{int} - \psi_{own}
\end{aligned}
\tag{IV.2}
$$

Verification tools are often not able to encode these equations. Thus, these functions are converted to ordinary differential equations (ODEs) by computing their time derivatives. Finally, these and the Dubins models are combined into a single nonlinear model with 9 state variables, as described in Eq. IV.3.

$$
\begin{aligned}
\dot{x}_1 &= \dot{x}_{own} = vcos(\psi_{own}), \\
\dot{x}_2 &= \dot{y}_{own} = vsin(\psi_{own}), \\
\dot{x}_3 &= \dot{\psi}_{own} = u, \\
\dot{x}_4 &= \dot{x}_{int} = vcos(\psi_{int}), \\
\dot{x}_5 &= \dot{y}_{int} = vsin(\psi_{int}), \\
\dot{x}_6 &= \dot{\psi}_{int} = c, \\
\dot{x}_7 &= \dot{\rho} = \frac{(y_{int} - y_{own})(\dot{y}_{int} - \dot{y}_{own}) + (x_{int} - x_{own})(\dot{x}_{int} - \dot{x}_{own})}{\sqrt{(x_{int} - x_{own})^2 + (y_{int} - y_{own})^2}}, \\
\dot{x}_8 &= \dot{\theta} = \frac{2(\dot{y}_{int} - \dot{y}_{own})(x_{int} - x_{own} + \rho) - 2(y_{int} - y_{own})(\dot{x}_{int} - \dot{x}_{own} + \dot{\rho})}{(y_{int} - y_{own})^2(x_{own} - x_{int} + \rho)^2} - \dot{\psi}_{own}, \\
\dot{x}_9 &= \dot{\psi} = \dot{\psi}_{int} - \dot{\psi}_{own}
\end{aligned}
\tag{IV.3}
$$

ACAS Xu NN input ranges require that θ and ψ must be in the $[-\pi, \pi]$ interval. Thus, it is necessary to ensure that the angles are always defined in the $[-\pi, \pi]$ range for each control step.

IV.3.1 NNCS benchmark

Combining the ACAS Xu neural network system and the nonlinear dynamical model, the ACAS Xu NNCS benchmark is presented in a diagram in Fig. IV.5. There are four main components or operations highlighted:

- 1: plantReach(): The plant dynamics are simulated based on a specified initial state (initSet) and advisory (adv_{NN}) for two seconds (sampling time).
- 2: normalizeNNinputs(): Then, the outputs of the plant (outPlant) are scaled and normalized.
- 3 and 4: NNreach():

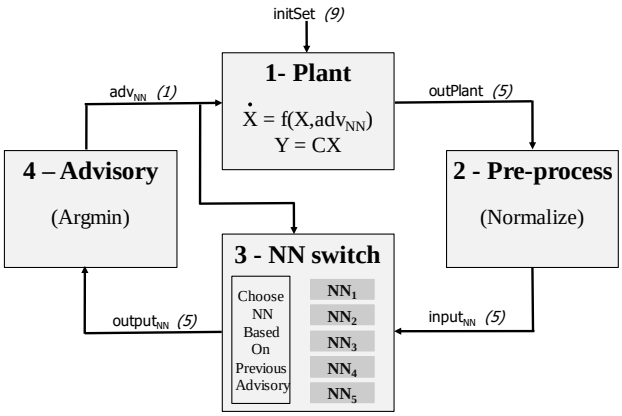


Figure IV.5: Diagram of the ACAS Xu neural network closed loop system.

- 3:Reach(): The scaled outputs (input$_{NN}$) are processed by the corresponding neural network controller, which is selected based on the previous advisory and the loss of vertical separation, as described in Section IV.2.2.
- 4: argminNN(): Based on the five outputs of the neural network (output$_{NN}$), the turn rate advisory (adv$_{NN}$) is computed (argmin) and commanded to the plant.

The name of the variables being processed by the next component are defined on top of the arrows and their respective dimensions are specified in parentheses, i.e., outPlant (5) is a five dimensional variable output in 1) and normalized in 2). It is assumed that the velocities and the flight altitude of both aircraft is constant. Thus, the ACAS Xu neural network system is able to choose between the five neural networks corresponding to no time to loss of vertical separation ($\tau = 0$). The sampling time of 2 seconds is selected based on the dynamics of the airplane, which doing it too often and changing the direction of the trajectory may be chattering, while doing it too infrequently may lead to crashes.

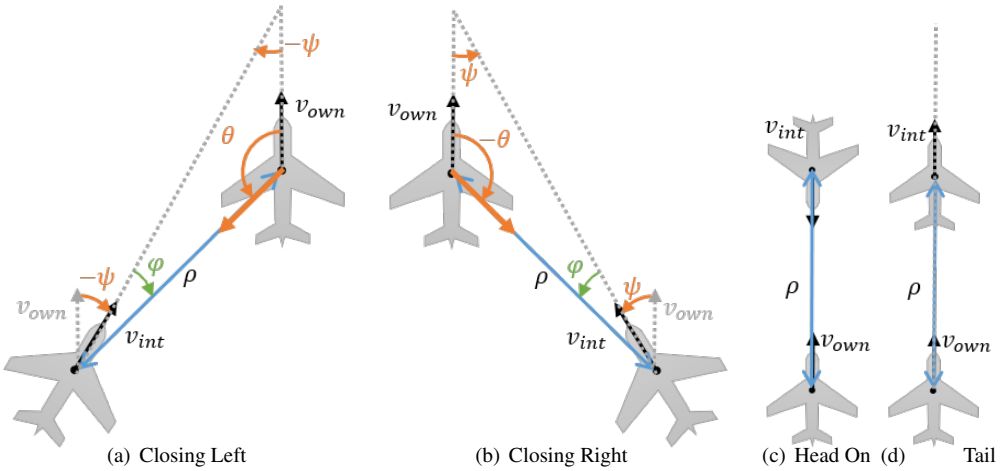
IV.4 Closed Loop Verification Test Case Generation

The objective of the closed loop verification tests is to demonstrate that the ownship and intruder aircraft remain safely separated given a set of initial conditions with uncertainty. The benchmarks presented here are not intended to provide complete coverage of the state space, but rather provide a challenge for NNCS verification approaches with a collection of 10 closed loop properties or test cases defined in Table IV.5 and depicted in Fig. IV.7. The closed loop properties presented in this work are a complement to the open loop properties, both of which sample portions of the state space. The closed loop cases specifically sample from a set of cases on a collision course, while the open loop cases do not specifically look at collision cases. A larger sampling outside of the 10 individual test cases is created by adding uncertainty about these points

(for example ±5000 ft in x and ± 200 ft in y). Then the set of states reachable from the initial set of states is found using reachability tools such as NNV or nnenum. These benchmarks are not intended to provide complete coverage of the state space, but rather demonstrate the capabilities of star set-based verification approach, a more comprehensive approach with formal guarantees, as an alternative to Monte Carlo or Design of Experiments-based simulation analysis.

For the closed loop benchmarks, the unsafe set is defined by the NMAC cylinder when there is a separation of less than 100 ft vertically or 500 ft horizontally. To scope the testing, the following ranges of variables are used: distances from 0 to the maximum turn diameter ($\rho \in [0, 87472]$ ft), the full range of angles to the intruder and heading of the intruder with respect to the ownship ($\theta \in [-\pi, \pi], \psi \in [-\pi, \pi]$), and the full range of reasonable velocities from a slow takeoff velocity to over Mach 1 ($v_{int} \in [60, 1145]$ ft/s, $v_{own} \in [100, 1145]$ ft/s). These limits, including different velocity ranges for the ownship and intruder, are inspired by previous work in defining open loop verification properties [130]. By law of sines, the variables used to describe the relative aircraft geometry are related with the time to collision T as described in Eq. IV.4, and four examples (closing left, closing right, head on and tail chase) are depicted in Fig. IV.6. Any collision test case can be generated by fixing the time to collision T, θ, v_{int}, and ρ, and computing the remaining variables as described in Table IV.4.

$$\frac{\rho}{\sin\psi} = \frac{v_{own}T}{\sin(\theta - \pi - \psi)} = \frac{v_{int}T}{sin(2\pi - \theta)} \tag{IV.4}$$



(a) Closing Left (b) Closing Right (c) Head On (d) Tail Chase

Figure IV.6: Closed Loop Test Case Geometry Examples

To generate a standard set of closed loop test cases, θ, v_{int}, and ρ were selected from the following discretized sets of values and used to calculate appropriate values of v_{own} and ψ: $\theta \in$ {-3π/4,-π/2,-

Table IV.4: Computing Ownship Velocity and relative heading from randomly selected angle to intruder, intruder velocity, and distance to the intruder

	Closing Left	Closing Right	Head On	Tail Chase
θ	$0 < \theta < \pi$	$-\pi < \theta < 0$	$\theta = \pm\pi$	$\theta = 0$
ψ	$-\sin^{-1}\left(\frac{\rho}{Tv_{int}} sin(\lvert\theta\rvert)\right)$	$\sin^{-1}\left(\frac{\rho}{Tv_{int}} sin(\lvert\theta\rvert)\right)$	0	0 or π
φ (from sum of angles)	$\pi - \lvert\theta\rvert - \lvert\psi\rvert$	$\pi - \lvert\theta\rvert - \lvert\psi\rvert$	-	-
v_{own}	$\frac{v_{int}\sin(\varphi)}{sin(\lvert\theta\rvert)}$	$\frac{v_{int}\sin(\varphi)}{sin(\lvert\theta\rvert)}$	$\frac{Tv_{int}-\rho}{T}$	$\frac{Tv_{int}+\rho}{T}$

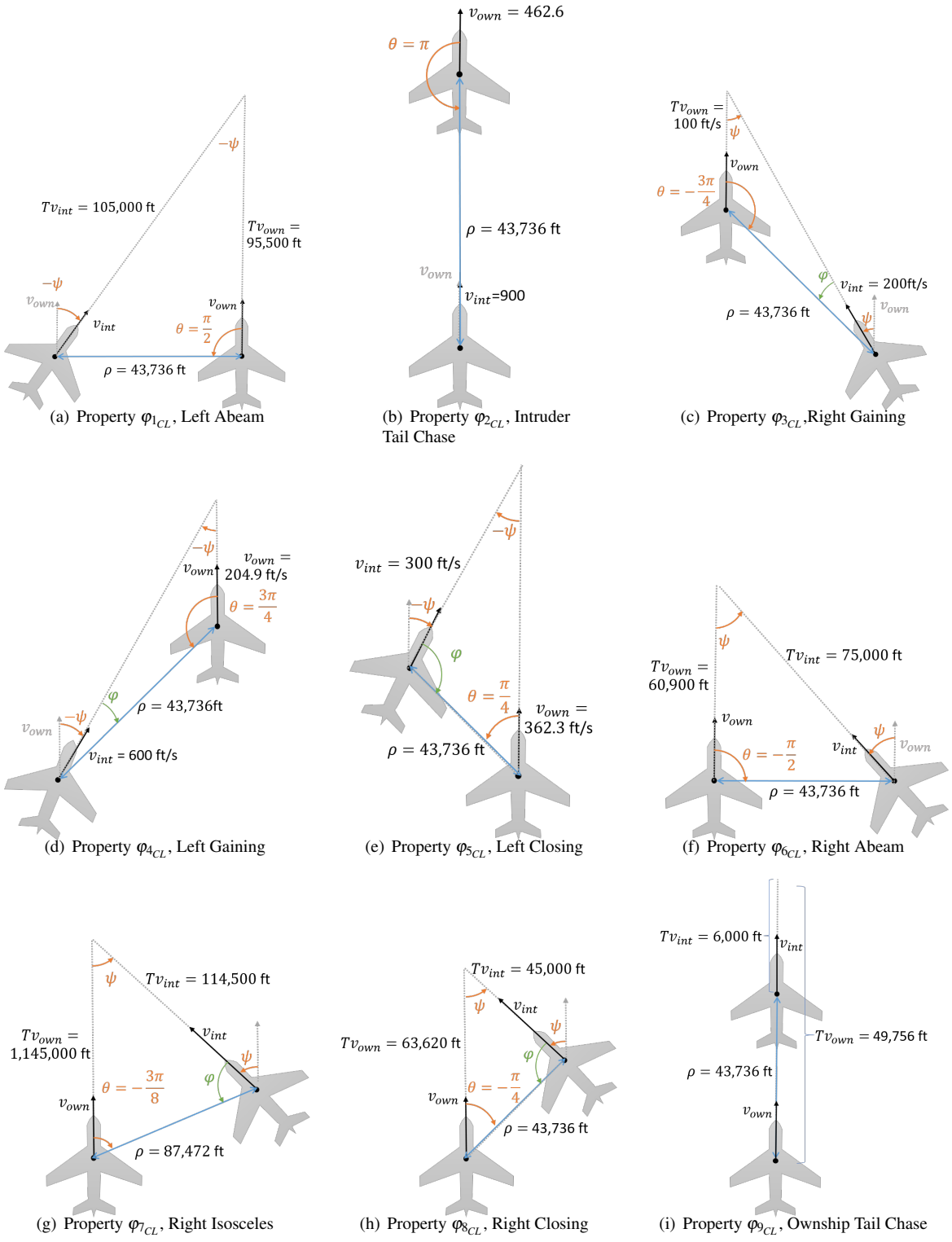
$3\pi/8$, -$\pi/4$, 0, $\pi/4$, $\pi/3$, $3\pi/4$, π } rad, $v_{int} \in \{60, 150, 300, 450, 600, 750, 900, 1050, 1145\}$ ft/s, and $\rho \in \{10000, 43736, 87472, 120000\}$ ft. Using a Latin hypercube sampling of v_{int} and θ for baseline coverage, the test cases in Table IV.5 may be used to evaluate a range of properties across the state space. To compute the Cartesian coordinates of the intruder, Eqs. IV.5-IV.6 are used. With the exception of the head on collision test case 10, represented by Fig. IV.6(c), each of the test cases are selected so that both aircraft are traveling in roughly the same direction, as depicted in Fig. IV.7. This is because aircraft are generally separated by at least 1000 feet in altitude when traveling in opposite directions, as commanded by Air Traffic Control or federal regulations. For example, the Code of Federal Regulations, Title 14, Section 91.159 instructs pilots to fly at an odd altitude (plus 500 feet for pilots operating under visual flight rules) when traveling East on a heading between 0-179 degrees, or an even altitude (plus 500 feet for pilots operating under visual flight rules) when traveling West on a heading between 180-359 degrees [140]. This work did not consider non-straight trajectories of the ownship and intruder, which would need to be considered in a more comprehensive safety assessment.

$$x_{int} = -\rho \sin(\theta) \tag{IV.5}$$

$$y_{int} = \rho \sin(\pi/2 - \theta) \tag{IV.6}$$

IV.5 Analysis Approaches

This section describes the fundamental theory behind star sets used to approximate reachable sets, as well as the closed loop neural network verification approaches. In terms of complexity, the reachability analysis of nonlinear ODEs is undecidable [141], NN reachability problem is NP-complete [130, 142], thus the reachability analysis of the NNCS with nonlinear ODE plant is undecidable.



(a) Property $\varphi_{1_{CL}}$, Left Abeam

(b) Property $\varphi_{2_{CL}}$, Intruder Tail Chase

(c) Property $\varphi_{3_{CL}}$,Right Gaining

(d) Property $\varphi_{4_{CL}}$, Left Gaining

(e) Property $\varphi_{5_{CL}}$, Left Closing

(f) Property $\varphi_{6_{CL}}$, Right Abeam

(g) Property $\varphi_{7_{CL}}$, Right Isosceles

(h) Property $\varphi_{8_{CL}}$, Right Closing

(i) Property $\varphi_{9_{CL}}$, Ownship Tail Chase

Figure IV.7: Depiction of the aircraft encounter geometry for the closed loop ACAS Xu verification properties

Table IV.5: ACAS Xu Benchmark Closed Loop Test Cases

Test Point	Name	v_{int} (ft/s)	θ (rad)	ρ (ft)	v_{own} (ft/s)	ψ (rad)	x_{int} (ft)	y_{int} (ft)
$\varphi_{1_{CL}}$	Left Abeam	1050	$\pi/2$	43,736	954.6	-0.4296	-43,736	0
$\varphi_{2_{CL}}$	Intruder Tail Chase	900	π	43,736	462.6	0	0	-43,736
$\varphi_{3_{CL}}$	Right Gaining	200	$-3\pi/4$	43,736	100	0.3617	30,926	-30,926
$\varphi_{4_{CL}}$	Left Gaining	600	$3\pi/4$	43,736	204.9	-0.5415	-30,926	-30,926
$\varphi_{5_{CL}}$	Left Closing	300	$\pi/4$	43,736	362.3	-0.2379	-30933	30933
$\varphi_{6_{CL}}$	Right Abeam	750	$-\pi/2$	43,736	609.3	0.6226	43,736	0
$\varphi_{7_{CL}}$	Right Isosceles	1145	$-3\pi/8$	87,472	1145.9	0.7835	80,814	33,474
$\varphi_{8_{CL}}$	Right Closing	450	$-\pi/4$	43,736	636.2	0.7577	30,926	30,926
$\varphi_{9_{CL}}$	Ownship Tail Chase	60	0	43,736	497.4	0	0	43,736
$\varphi_{10_{CL}}$	Head On Collision	600	0	120,000	600	π	0	120,000

IV.5.1 Star Set Theory

A star set [72, 108] (or simply star) Θ is a tuple $\langle \kappa, V, P \rangle$ where $\kappa \in \mathbb{R}^n$ is the center vector, $V = \{\mu_1, \mu_2, \cdots, \mu_m\}$ is a set of m basis vectors in $\mathbb{R}^n$, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate. The set of states represented by the star is given as:

$$[\![\Theta]\!] = \{\omega \mid \omega = \kappa + \Sigma_{i=1}^{m}(\alpha_i \mu_i) \text{ such that } P(\alpha_1, \cdots, \alpha_m) = \top\}. \tag{IV.7}$$

Sometimes both the tuple Θ and the set of states $[\![\Theta]\!]$ are referred to as Θ. In this work, the predicates are restricted to be a conjunction of linear constraints, $P(\alpha) \triangleq H\alpha \leq d$ where, for p linear constraints, $H \in \mathbb{R}^{p \times m}$, α is the vector of m-variables, i.e., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. A star is an empty set if and only if $P(\alpha)$ is empty.

Any convex polyhedron can be represented as a star set. An affine mapping of a star set $\Theta = \langle \kappa, V, P \rangle$ defined by a mapping matrix W and an offset vector b is another star set $\Theta' = \langle W\kappa + b, WV, P \rangle$. The intersection of a star set $\Theta = \langle \kappa, V, P \rangle$ with a polyhedron $G \triangleq H_G\alpha \leq d_G$ is another star set $\Theta' = \langle \kappa, V, P \wedge P_G \rangle$, where $P_G \triangleq H_G \times V\alpha \leq d_G - H_G \times \kappa$.

IV.5.2 Neural Network Verification (NNV) Tool

The NNV[1] tool is one of the two verification tools used to analyze the safety and functionality of the ACAS Xu NN system. This tool is evaluated on the ten closed-loop benchmarks where the safety of the ownship is analyzed with respect to one intruder aircraft. The analysis consists of evaluating, under multiple scenarios and initial state uncertainty, the safety of the ownship aircraft using Algorithm 2, which follows the general computation flow to the one described in section IV.3.1. For the nonlinear plant reachability analysis, NNV makes use of a CORA [65] integration.

The sets are represented using star-sets, and the reachability functions compute over-approximations of

[1] https://github.com/verivital/nnv

Algorithm 2: ACAS Xu NNCS reachability algorithm in NNV

```
Result:
Rs: Plant reachable states
Initialize
initSet // Initial state set
a_prev // Previous advisory
tF // Number of control steps
safe = True // Safety variable, change to false if NMAC violated
v_int // Intruder velocity
v_own // Ownship velocity
while safe do
    for (i = 0; i < tF; i = i + 1) do
        [state_set,outPlant] = plantReach(initSet , a_prev) // reachability analysis of the plant
        ρ = extractDist(state_set)// Extract distance set
        if ρ ¡ 500 then
            safe = False
        end
        input_NN = normalizeNNinputs(outPlant) // normalize inputs to the neural network
        adv_NN = NNreach(a_prev, input_NN) // reachability analysis of the Nueral Network (NN) controller, see Algorithm 3.
        initSet = state_set
        a_prev = adv_NN
        Rs[i] = state_set
    end
end
return: Rs
```

the reachable sets. Due to the initial uncertainty added to the initial states of the aircraft, it is possible that multiple advisories are issued at each time step, increasing the number of possible trajectories of the ownship. In this case, the reach sets are split and paired with the corresponding advisory to decrease the number of operations and trajectories computed in order to reduce the conservativeness and memory consumption. The workflow of this algorithm implemented in NNV using star-set methods is presented in Algorithm 3. The importance of this algorithm is illustrated with the following example with one initial advisory and one initial state set, where the number of operations is exponentially reduced. For the purpose of this example, it is assumed that at each time step, there are 2 possible advisories (output_{NN}) issued for each plant output (output_{P}) set computed.

EXAMPLE 1.

Step 1) There is **1** output_{P} set computed based on the initial state and previous advisory. For this output_{P} set, there are 2 possible advisories (output_{NN}) computed by a NN. *Step 2)* For each advisory, one output_{P} is computed based on the initial state, obtaining **2** sets. For each possible combination of advisories and output_{P} sets (4), there are 2 advisories issued, increasing the number of advisories to 8. However, since there is a limit of five maximum unique advisories (*COC,WL,WR,SL,SR*), these 5 advisories are utilized for the remainder of the example. *Step 3)* There are 5 advisories and 2 state sets, increasing the number of plant output_{P} sets

to **10** in step 3. By computing all possible combinations between the 10 output$_P$ sets and 5 advisories, there are a total of 50 NN output$_{NN}$ sets to compute, although the number of advisories computed is limited to 5 as previously described. *Step 4)* There are now 10 initial sets and 5 advisories, out of which **50** plant output$_P$ sets are computed. Following this procedure, it is observed that the number of output$_P$ sets computed grows by a multiple of 5 at each future control step, i.e., in *Step 5)* there are **250** plant output$_P$ sets, in *Step 6)* there are **1250** sets and so on.

EXAMPLE 2.

The number of plant output$_P$ sets can be significantly reduced by using Algorithm 3. *Step 1)* There is **1** output$_P$ set, for which there are 2 possible advisories computed by a NN. *Step 2)* Based on the initial state set, for each of the 2 advisories the output$_P$ set of the plant is computed, obtaining **2** sets. By tracking which set corresponds to which advisory, the number of operations are reduced from 4 to 2. For each combination, there are 2 advisories issued, increasing the number of advisories to 4. *Step 3)* There are 4 advisories and 2 output$_P$ sets. For each advisory, the corresponding plant output$_P$ set is computed for a total of **4** plant output$_P$ sets. Then, for each output$_P$ set, there are 2 advisories issued for a total of 8 advisories. *Step 4)*. The relationship between the advisories and the plant output$_P$ sets computed is one to one, therefore **8** sets are obtained. Following this pattern, it is observed that the number of plant output$_P$ sets increases by 2 for each control step, i.e. in *Step 5)* there are **16** plant output$_P$ sets, in *Step 6)* there are **32** sets and so on. If both examples are run for 10 control steps, the total number of output$_P$ sets are reduced from **781250** to **512**.

Algorithm 3: Reachability algorithm of switching classification based controllers — *NNreach*

Result:
adv$_{NN}$: Advisories paired with each plant reach set
Inputs
a_{prev} // Previous advisory
$input_{NN}$ // inputs to NNs
N // number of reach sets & previous advisory pairs
k = 0
for $(i = 0;\ i < N;\ i = i + 1)$ **do**
    output$_{NN}$[i] = Reach(input$_{NN}$[a$_{prev}$[i,1]] , a$_{prev}$[i,0]) // Compute reach set of neural network
    advise[i] = argminNN(output$_{NN}$[i]) // Compute advisory based on minimum value of output reach set output$_{NN}$
    for *adv in advise[i]* **do**
        adv$_{NN}$ [k] = [*adv*, i] // Track advisories
        k++
    end
end
return: adv$_{NN}$

IV.5.3 Neural Network Enumeration (nnenum) Tool

The second verification tool used for analysis is nnenum[2], which uses star sets to reason over possible neural network outputs given sets of input states. Nnenum combines many engineering improvements to the base star set method [143], such as using quick bounds estimates using zonotopes with domain contraction to improve accuracy. This is further combined with an abstraction-refinement approach [127], where star sets first analyze the system with over-approximation, and then refine by splitting sets on individual neurons only if it is needed to prove the safety property.

In this work, first it is analyzed a set of states using the quicker over-approximation mode of nnenum to check whenever multiple advisories are possible. Otherwise, exact analysis is used, which splits the set of states into many smaller states, each with a unique classification. For the plant dynamics, rather than using CORA and the nonlinear differential equations in Equation IV.3, nnenum instead performs numerical simulation within each set using the original Dubin's car dynamics in Equation IV.1 and observations from Equation IV.2, in order to produce a local numerical linearization of the observed state variables. Since reachability analysis through linear differential equations is more efficient than reachability with nonlinear equations, the method has better expected scalability. This result is demonstrated in our evaluation section next, where larger uncertainties in the initial state can be checked, with accuracy that visually resembles simulations of the closed-loop system.

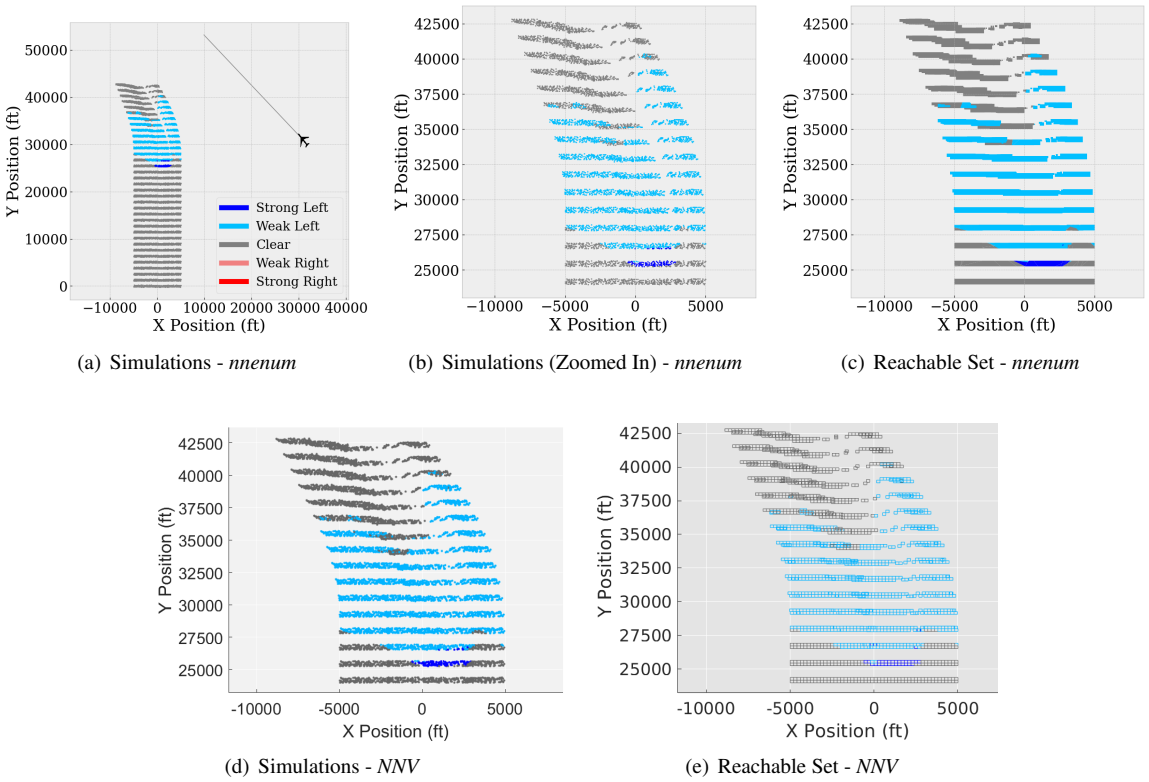
IV.6 Results

The 10 closed loop test cases depicted on Fig. IV.7 are evaluated using NNV and nnenum under specified initial uncertainty. For each scenario, an initial ownship uncertainty of ± 5000 in the x direction and ± 200 in the y direction is used when evaluated. Although there are some small differences in 2 out of the 10 cases evaluated between NNV and nnennum ($\varphi_{2_{CL}}$ and $\varphi_{10_{CL}}$), both tools are able to succesfully verify the safety of the ownship aircraft with respect to the intruder in all 10 cases.

These results are depicted in ten figures, which consist of 5 subplots, organized as follows: subplots *a*, *b* and *c* correspond to nnenum results and subfigures *d* and *e* to NNV. For each tool there are zoomed in (*b* and *d*) simulations results as well as the reachable sets (*c* and *e*), while *a* depicts the complete simulation trajectories of the ownship in nnenum.

To illustrate and explain the results from this investigation, 2 out of the 10 test cases are visualized in this section: Right Closing $\varphi_{8_{CL}}$ (Fig. IV.8) and Head On Collision $\varphi_{10_{CL}}$ (Fig. IV.9). The remaining results are found in Figs. IV.10 to IV.17. This set of results can be summarized as the ownship is always safe, maintaining a much greater distance than the safety distance specified in the NMAC IV.1. This is accomplished by turning

[2]https://github.com/stanleybak/nnenum



(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

(d) Simulations - *NNV* (e) Reachable Set - *NNV*

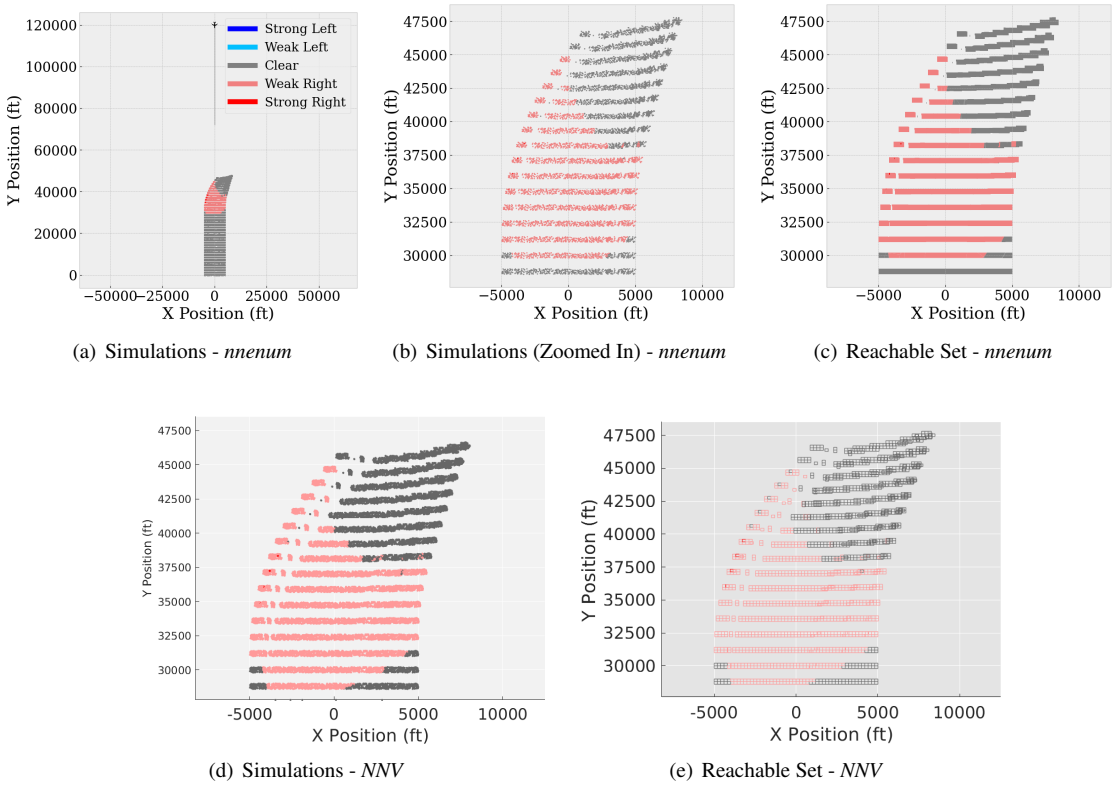
Figure IV.8: Right Closing $\varphi_{8_{CL}}$ with initial ownship uncertainty x_0, y_0 = [±5000, ±200].

to the opposite side from which the intruder is approaching as depicted in Figs. IV.8 and IV.10, and Figs. IV.12 to IV.16. In the other 3 scenarios, the ownship is biased to deviate towards the right in order to avoid a collision with the intruder when both are flying in the same path but opposite direction or same direction and different speed. This bias is very evident in Fig. IV.9, and it is also observed in Fig. IV.17, in which the ownship reachable sets split into two main paths, one towards the rights, and one towards the left.

If the ownship is initially located between approx. -2000 ft and +5000 in the x-axis, the ownship will deviate to the right. If initially located to the left of -2000 ft in the x-axis, then the ownship will avoid the collision by turning to the left. The final scenario, Intruder Tail Chase $\varphi_{2_{CL}}$, does not present this bias and behaves more as initially predicted. This result is demonstrated in Fig. IV.11 by both NNV and nnenum, although the tools present slightly different reachable set trajectories. The initial state set is split into two halves at x = 0 ft in both simulations and reachable sets. If the ownship is initially located in the positive x-axis, it will turn to the right and vice versa.

The first test case, Right Closing $\varphi_{8_{CL}}$, is illustrated in Fig. IV.8. It is observed in the simulation plots that when the ownship approaches the intruder's trajectory at approximately y = 25000 ft, the controllers start issuing weak and strong left commands to the ownship to maintain a safe distance with the intruder. These

decisions are specially evident in the zoomed in and reach set plots, where the trajectories turn towards the left for a few steps, followed by clear of conflict commands as the ownship safety is maintained. On the other scenario in Fig. IV.9, the Head On Collision $\varphi_{10_{CL}}$ test case demonstrates the ownship's safety when the ownship faces another aircraft in the same path but in the opposite direction. Both tools show very similar results as the ownship deviates to the right when flying at approximately y = 30000 ft, issuing only weak right commands to maintain safety with respect to the intruder.



(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

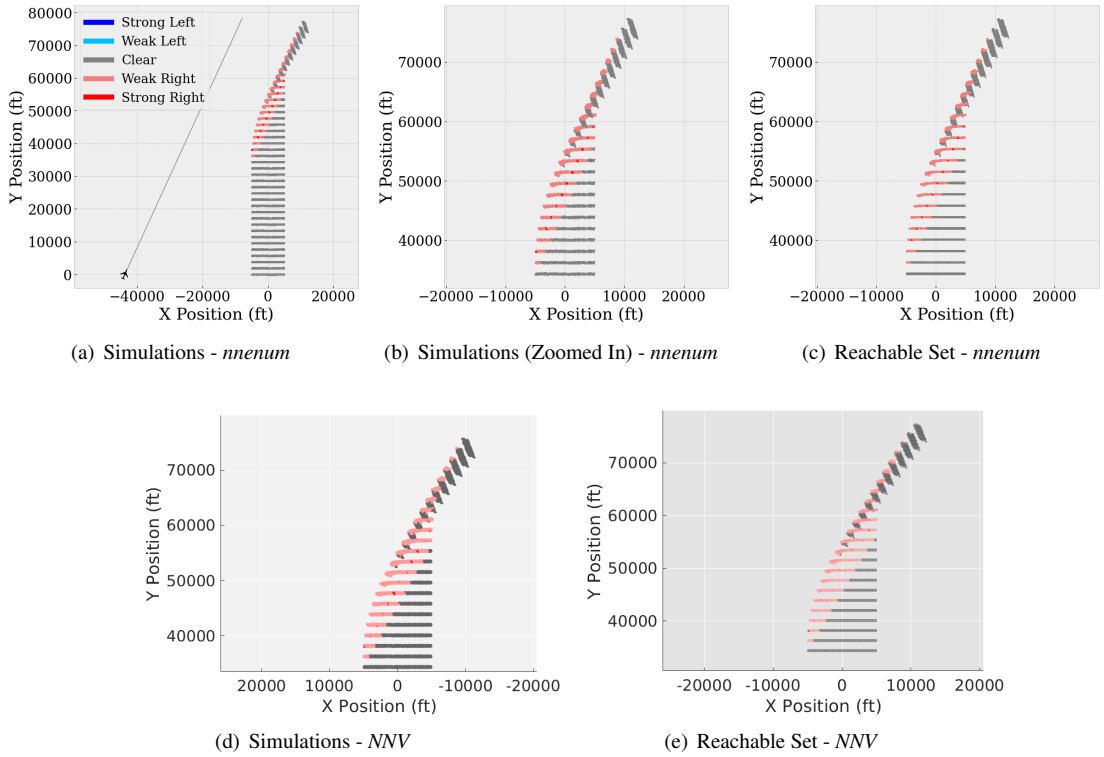
(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.9: Head On Collision $\varphi_{10_{CL}}$ with initial ownship uncertainty x_0, y_0 = [±5000, ±200].

The similarity between the simulation and verification results observed in these plots is an indication of the reduced over-approximation errors computed by both tools using star-set methods. Despite both tools successfully verifying the safety of the ownship with respect to the intruder aircraft in all 10 cases, there are some main differences between nnenum and NNV, discussed in the next section.

IV.7 Discussion

The results in Section IV.6 demonstrate the capabilities of nnenum and NNV for the verification of real-world unmanned CPSs, where it can also be observed some of the main differences between these tools. NNV needs to compute an initial partition on the initial set and then compute the reachable sets for each partition, while nnenum is able to compute these partitions as needed at each control step, improving its



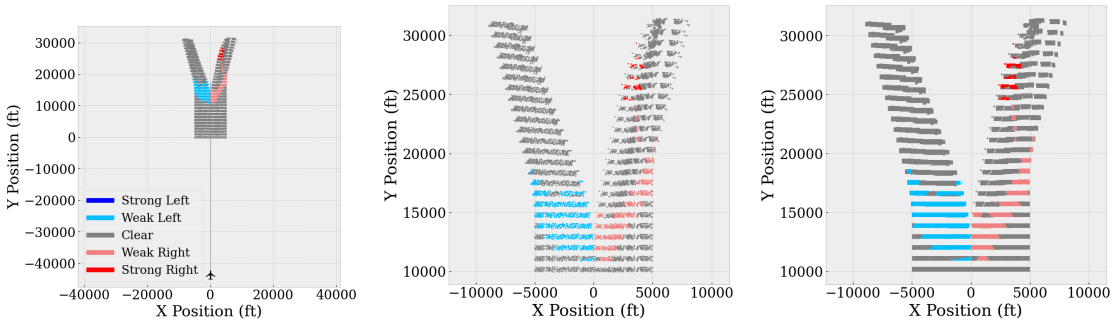
(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

(d) Simulations - *NNV* (e) Reachable Set - *NNV*

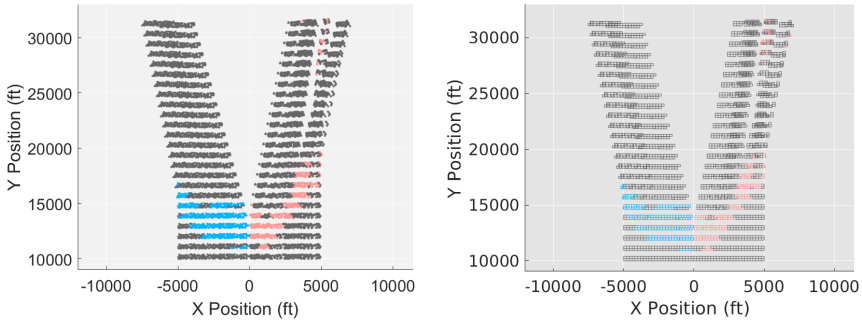
Figure IV.10: Left Abeam $\varphi_{1_{CL}}$ with initial ownship uncertainty x_0, $y_0 = [\pm 5000, \pm 200]$.

efficiency with respect to NNV. Another main difference is the nonlinear ODE reachability analysis, where NNV is also capable of computing and displaying the complete continuous-time reachable sets of NNCS, but for consistency purposes, only the reachable sets of the ownship at each control step are shown for ease of comparison against nnenum. The results also show some of the complexities of deploying and verifying these complex systems. In 8 out of 10 cases, NNV and nnenum obtain the same results, but there are two cases in which there are differences in both the simulations and the reachability analysis. There is a specific step in each scenario in which NNV and nnenum do not issue the same advisory commands. In test case $\varphi_{10_{CL}}$ (Fig. IV.9), the main difference is in the first reachable set shown, in which nnenum only selects COC commands while NNV also has some weak right commands towards the center of the reachable set. The other test case, $\varphi_{2_{CL}}$ (Fig. IV.11) seems similar to the previous test case in the sense that there is one main step in the reachable set computation that differs between NNV and nnenum (step 5) in both sides of the path, which leads to different final paths. To complement the reachability analysis presented and try to better understand the differences between NNV and nnenum, the simulation differences between NNV and nnenum have been analyzed across 1000 randomly initialized simulations and the results are introduced in Table IV.6.

It is observed that even in the test cases in which NNV and nnenum return the same results, there are still



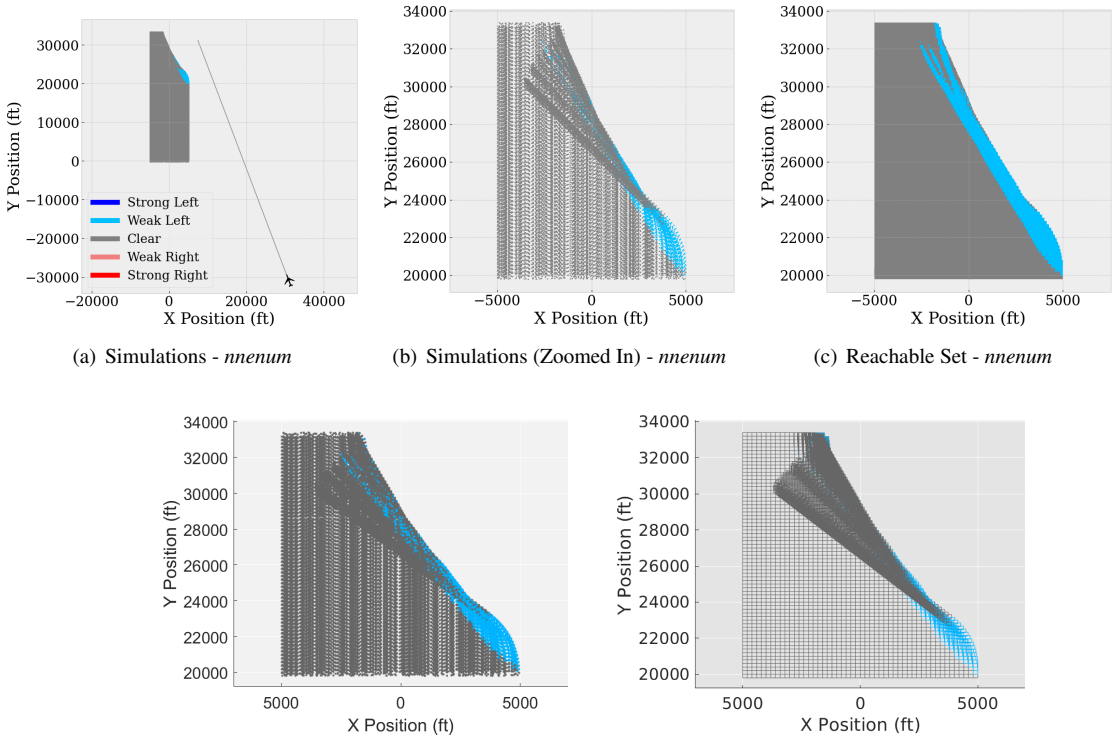
(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*



(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.11: Intruder Tail Chase $\varphi_{2_{CL}}$ with initial ownship uncertainty $x_0, y_0 = [\pm 5000, \pm 200]$.

some minor differences between the two. Some possible sources of error are the different solvers used for the computation of the states of the plant, or a slightly different implementation on the plant model, among others. NNV computes the reachability analysis of the plant model using all 9 state variables described in Eq. IV.3, but nnenum makes use of only the first 6 variables, and then uses those to compute the remaining 3 (7 to 9) a posteriori. These differences in computation may lead to small computation differences between the two, which accumulate through time and eventually lead to slightly different final paths, although both tools successfully verify the safety of the ownship in every case. Another main difference between NNV and nnenum is the different reachability schemes utilized, which lead to significant differences in the computation time. In average, nnenum is able to compute the reachable set of each closed loop property in 172 seconds, while NNV completes each test case in 140819 seconds, approximately 39 hours. The main reasons for the difference in computation time are the different partitioning routines and the plant reachability techniques. When compared to other validation techniques such as Monte Carlo simulations, these reachability techniques can prove and guarantee the safety of the ownship under the whole initial uncertainty of the ownship aircraft, while Monte Carlo simulations can only provide probable guarantees, determined by the number of simulations run. In terms of computation time, nnenum is equivalent to running 0.13 simulations



(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

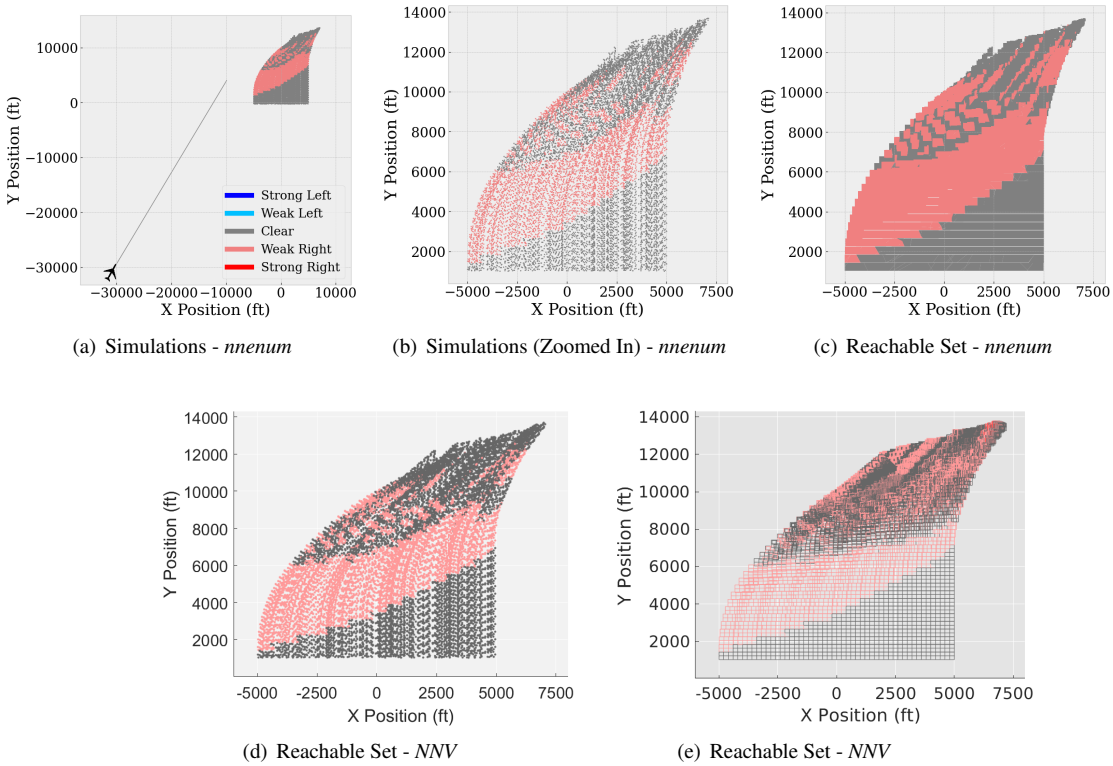
(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.12: Right Gaining $\varphi_{3_{CL}}$ with initial ownship uncertainty x_0, $y_0 = [\pm 5000, \pm 200]$.

per sqft of the initial set of the ownship, while NNV is equivalent to running 114 simulations per sqft. In this aspect, nnenum is preferable over Monte Carlo simulations and NNV due to its efficient computation of the reachable sets of this ACAS Xu closed-loop benchmark.

IV.8 Summary

This chapter introduced a set of 10 new closed loop verification properties and developed and applied a closed loop verification approach to verify both the output of NNCS and the switching behavior between neural network controllers. The approach was demonstrated using the NNV and nnenum tools on 5 of 45 ACAS Xu neural networks with switching between all five corresponding to co-altitude collision cases. Both tools show reachable states of an ownship aircraft with modified Dubins dynamics on a collision course with an intruder aircraft. The reachability computation considers initial state uncertainty and ownship control inputs provided by 5 switched NNs. The verification objective was to show that the switch NNCS resulted in collision free courses, even a large initial uncertainty. This study showed that it is possible to do a comprehensive safety verification analysis of a complex air collision advisory system for aircraft travelling in straight, co-altitude paths, but further research is needed to determine if climbing or descending flights can also be proven safe



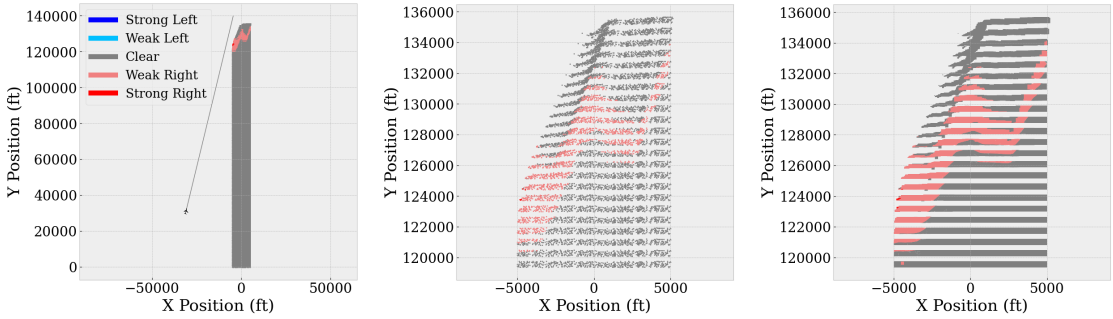
(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

(d) Reachable Set - *NNV* (e) Reachable Set - *NNV*

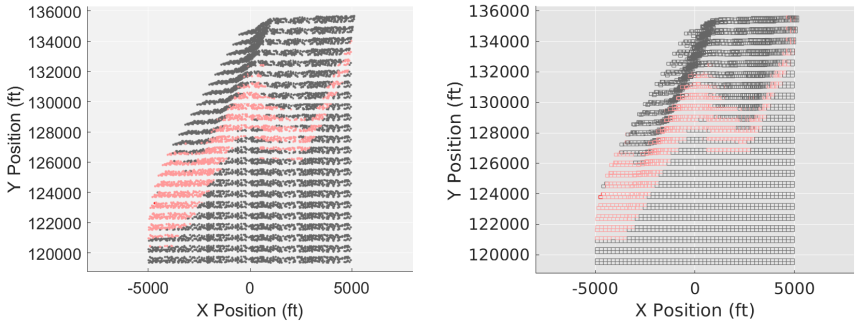
Figure IV.13: Left Gaining $\varphi_{4_{CL}}$ with initial ownship uncertainty x_0, $y_0 = [\pm 5000, \pm 200]$.

using these tools. In the tool demonstration, nnenum showed a tighter and faster computation of the over-approximation of its reachability methods, as observed by the similarity between the simulation and reachable plots, while NNV is able to present the complete continuous-time reachable sets of the ownship trajectories. A summary of the main outcomes of our methods can be summarized as follows:

- For an initial assessment, a small amount of Monte Carlo simulations may be sufficient, in combination of other testing routines, to check the proper behavior of the CPS and its implementation.
- In safety-critical applications, Monte Carlo simulations cannot provide formal guarantees, which is vital for safety-critical systems such as autonomous aircraft. In this case, formal verification tools such as nnenum and NNV would be needed.
- NNV and nnenum can both provide formal guarantees on the safety of the aircraft using reachability analysis for neural networks as well as nonlinear ODEs.
- nnenum has proven to compute the reachable sets of the NNCS much faster than NNV, about 4 orders of magnitude faster, largely due to ts efficient partitioning scheme and the nonlinear ODE reachability method.

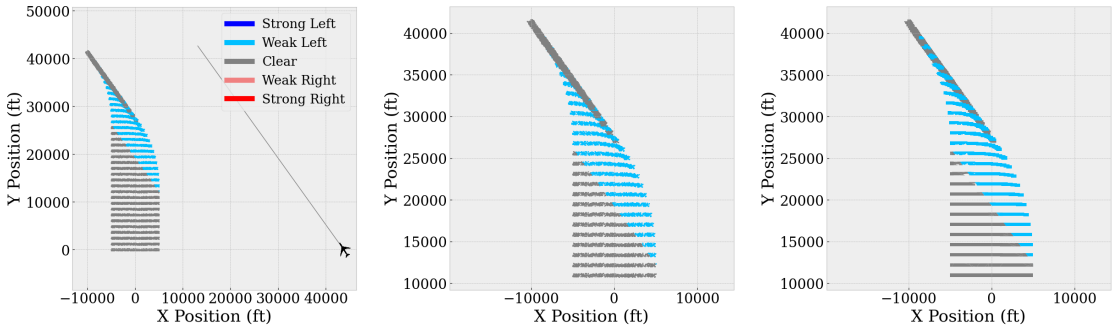


(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

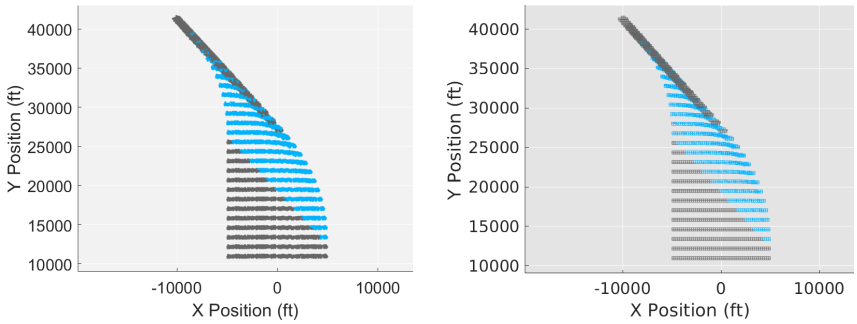


(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.14: Left Closing $\varphi_{5_{CL}}$ with initial ownship uncertainty $x_0, y_0 = [\pm 5000, \pm 200]$.

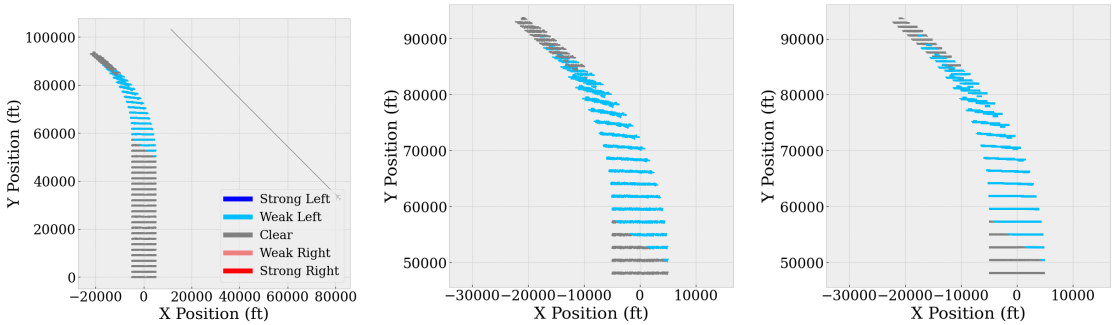


(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

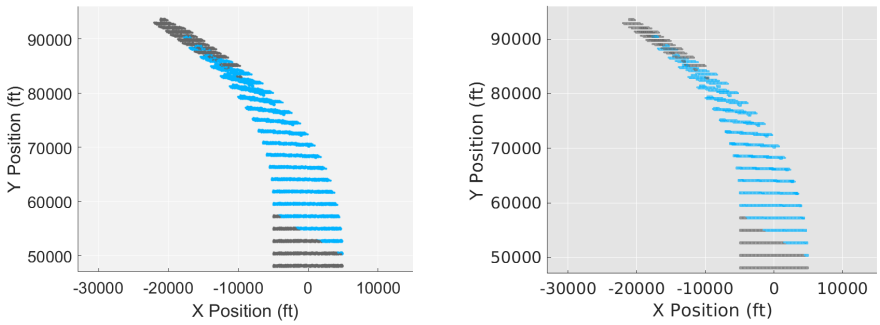


(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.15: Right Abeam $\varphi_{6_{CL}}$ with initial ownship uncertainty $x_0, y_0 = [\pm 5000, \pm 200]$.

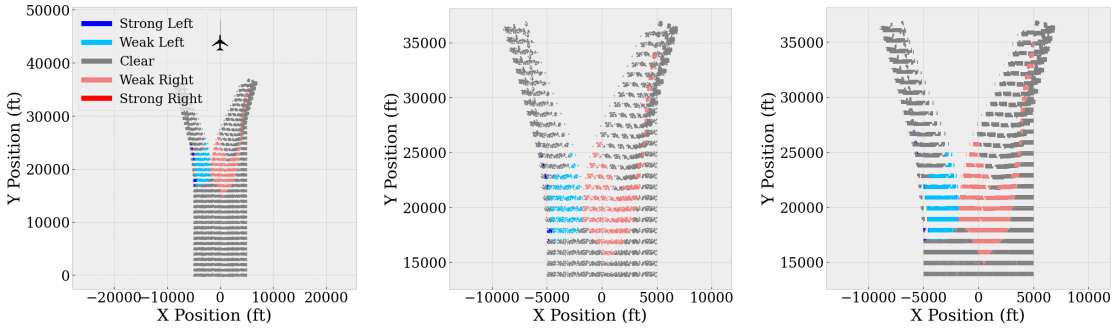


(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*

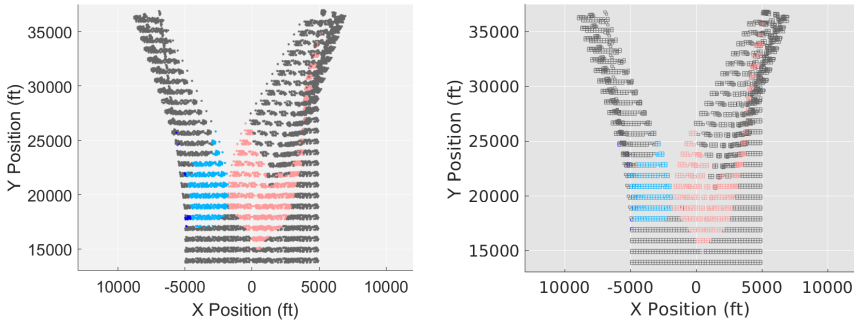


(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.16: Right Isosceles $\varphi_{7_{CL}}$ with initial ownship uncertainty x_0, y_0 = [±5000, ±200].



(a) Simulations - *nnenum* (b) Simulations (Zoomed In) - *nnenum* (c) Reachable Set - *nnenum*



(d) Simulations - *NNV* (e) Reachable Set - *NNV*

Figure IV.17: Ownship Tail Chase $\varphi_{9_{CL}}$ with initial ownship uncertainty x_0, y_0 = [±5000, ±200].

Table IV.6: Simulation differences between NNV and nnenum across all 10 closed-loop properties. For each experiment, 1000 random simulations were sampled. x_{own} and y_{own} values are in ft and ψ_{own} in rad, and they correspond to the average values of the absolute difference between the recorded variables in NNV and nnenum.

		Closed-Loop Properties									
		$\varphi_{1_{CL}}$	$\varphi_{2_{CL}}$	$\varphi_{3_{CL}}$	$\varphi_{4_{CL}}$	$\varphi_{5_{CL}}$	$\varphi_{6_{CL}}$	$\varphi_{7_{CL}}$	$\varphi_{8_{CL}}$	$\varphi_{9_{CL}}$	$\varphi_{10_{CL}}$
$\mathbf{x}_{own}$	*mean*	0.24	243.29	9.9e-11	7.4e-10	5.6e-11	1.4e-09	2.3e-9	2.1e-9	9.5e-10	71.86
	std	0.039	355.89	1.5e-10	5.6e-10	2.0e-10	1.5e-9	3.3e-9	2.8e-9	1.0e-9	105.41
	max	0.95	1.1e+3	3.7e-10	1.3e-9	8.8e-10	3.7e-9	8.9e-9	6.2e-9	2.3e-9	270.77
$\mathbf{y}_{own}$	*mean*	0.0562	51.58	4.8e-11	1.9e-10	1.3e-9	2.9e-10	7.3e-10	1.1e-10	1.2e-10	14.54
	std	0.098	75.94	7.7e-11	1.7e-10	1.5e-9	3.8e-10	1.3e-9	1.5e-10	1.3e-10	24.42
	max	0.26	241.61	2.7e-10	4.2e-10	4.7e-9	1.0e-9	3.9e-9	4.6e-10	3.4e-10	68.43
ψ_{own}	*mean*	1.2e-5	0.039	8.3e-16	6.3e-15	3.2e-16	2.6e-15	1.6e-15	2.0e-15	2.3e-15	0.0069
	std	3.2e-5	0.048	1.6e-15	4.9e-15	1.1e-15	2.8e-15	2.6e-15	3.0e-15	2.6e-15	0.011
	max	1.0e-4	0.11	5.7e-15	1.2e-14	4.9e-15	8.6e-15	8.5e-15	7.7e-15	5.7e-15	0.025

CHAPTER V

Verification of a General Class of Neural Ordinary Differential Equations

This chapter is adapted from the material presented in [144, 145].

Continuous deep learning models, referred to as Neural Ordinary Differential Equations (Neural ODEs), have received considerable attention over the last several years. Despite their burgeoning impact, there is a lack of formal analysis techniques for these systems. In this chapter, we consider a general class of neural ODEs with varying architectures and layers, and introduce a novel reachability framework that allows for the formal analysis of their behavior. The methods developed for the reachability analysis of neural ODEs are implemented in a new tool called NNVODE. Specifically, our work extends an existing neural network verification tool to support neural ODEs. We demonstrate the capabilities and efficacy of our methods through the analysis of a set of benchmarks that include neural ODEs used for classification, and in control and dynamical systems, including an evaluation of the efficacy and capabilities of our approach with respect to existing software tools within the continuous-time systems reachability literature, when it is possible to do so.

V.1 Introduction

Neural Ordinary Differential Equations (ODEs) were first introduced in 2018, as a radical new neural network design that boasted better memory efficiency, and an ability to deal with irregularly sampled data [146]. The idea behind this family of deep learning models is that instead of specifying a discrete sequence of hidden layers, we instead parameterize the derivative of the hidden states using a neural network [6]. The output of the network can then be computed using a differential equation solver [6]. This work has spurred a whole range of follow-up work, and since 2018 several variants have been proposed, such as augmented neural ODEs (ANODEs) and their ensuing variants [87, 84, 85]. These variants provide a more expressive formalism by augmenting the state space of neural ODEs to allow for the flow of state trajectories to cross. This crossing, prohibited in the original framework, allows for the learning of more complex functions that were prohibited by the original neural ODE formulation [84].

Due to the potential that neural networks boast in revolutionizing the development of intelligent systems in numerous domains, the last several years have witnessed a significant amount of work towards the formal analysis of these models. The first set of approaches that were developed considered the formal verifica-

tion of neural networks (NN), using a variety of techniques including reachability methods [14, 83, 147], and SAT techniques [130, 148]. Thereafter, many researchers proposed novel formal method approaches for neural network control systems (NNCS), where the majority of methods utilized a combination of NN and hybrid system verification techniques [58, 70, 55, 50, 51, 71]. Building on this work, a natural outgrowth is extending these approaches to analyze and verify neural ODEs, and some recent studies have considered the analysis of formal properties of neural ODEs. One such study aims to improve the understanding of the inner operation of these networks by analyzing and experimenting with multiple neural ODE architectures on different benchmarks [95]. Some studies have considered analyses of the robustness of neural ODEs such as [96] and [97], which evaluate the robustness of image classification neural ODEs and compare the efficacy of this class of network against other more traditional image classifier architectures. The first reachability technique targeted for neural ODEs presented a theoretical regime for verifying neural ODEs using Stochastic Lagrangian Reachability (SLR) [102]. This method is an abstraction-based technique that computes confidence intervals for the calculated reachable set with probabilistic guarantees. In a follow-up work, these methods were improved and implemented in a tool called GoTube [104], which is able to compute reach sets for longer time horizons than most state-of-the-art reachability tools. However, these methods only provide stochastic bounds on the reach sets, so there are no formal guarantees on the derived results.

To the best of our knowledge, this chapter presents the first deterministic verification framework for a general class of neural ODEs with multiple continuous-time and discrete-time layers. In this work, we present our verification framework NNVODE that makes use of deterministic reachability approaches for the analysis of neural ODEs. Our methods are evaluated on a set of benchmarks with different architectures and conditions in the area of dynamical systems, control systems, and image classification. We also compare our results against three state-of-the-art verification tools when possible. In summary, the contributions of this chapter are:

- We introduce a general class of neural ODEs that allows for the combination of multiple continuous-time and discrete-time layers.
- We develop NNVODE, an extension of NNV [14], to formally analyze a general class of neural ODEs using sound and deterministic reachable methods.
- We run an extensive evaluation on a collection of benchmarks within the context of time-series analysis, control systems, and image classification. We compare the results to Flow*, GoTube, and JuliaReach for neural ODE architectures where this is possible.

V.2 Background and Problem Formulation

Neural ODEs emerged as a continuous-depth variant of neural networks becoming a special case of ordinary differential equations (ODEs), where the derivatives are defined by a neural network, and can be expressed as:

$$\dot{z} = g(z), \qquad z(t_0) = z_0, \tag{V.1}$$

where the dynamics of the function $g : \mathbb{R}^j \to \mathbb{R}^p$ are represented as a neural network, and initial state $z_0 \in \mathbb{R}^j$, where j corresponds to the dimensionality of the states z. A *neural network* (NN) is defined to be a collection of consecutively connected $\mathcal{NN}$-Layers as described in Definition 2.

Definition 2 ($\mathcal{NN}$-Layer) *A $\mathcal{NN}$-Layer is a function h: $\mathbb{R}^j \to \mathbb{R}^p$, with input $x \in \mathbb{R}^j$, output $y \in \mathbb{R}^p$ defined as follows*

$$y = h(x) \tag{V.2}$$

where the function h is determined by parameters θ, typically defined as a tuple $\theta = \langle \sigma, \mathbf{W}, \mathbf{b} \rangle$ for fully-connected layers, where $\mathbf{W} \in \mathbb{R}^{j \times p}$, $\mathbf{b} \in \mathbb{R}^p$, and activation function σ: $\mathbb{R}^j \to \mathbb{R}^p$, thus the fully-connected $\mathcal{NN}$-Layer is described as

$$y = h(x) = \sigma(\mathbf{W}(x) + \mathbf{b}). \tag{V.3}$$

However, for other layers such as convolutional-type $\mathcal{NN}$-Layers, θ may include parameters like the filter size, padding, or dilation factor and the function h in (V.3) may not necessarily apply. For a formal definition and description of the reachability analysis for each of these layers that are integrated within NNVODE, we refer the reader to Section 4 of [149]. For the Neural ODEs (NODEs), we assume that g is Lipschitz continuous, which guarantees that the solution of $\dot{z} = g(z)$ exists. This assumption allows us to model g with m layers of continuously differentiable activation functions σ_k for $k \in \{1, 2, ..., m\}$ such as sigmoid, tanh, and exponential activation functions. Described in (V.1) is the notion of a NODE as introduced in [6] and an example is illustrated in Figure V.1.

Definition 3 (NODE) *A **NODE** is a function $\dot{z} = g(z)$ with m fully-connected $\mathcal{NN}$-Layers, and it is defined as follows*

$$\dot{z} = g(z) = h_m(h_{m-1}(...h_1(z))), \tag{V.4}$$

where $g : \mathbb{R}^j \to \mathbb{R}^p$, $\sigma_k : \mathbb{R}^{j_k} \to \mathbb{R}^{p_k}$, $\boldsymbol{W}_k \in \mathbb{R}^{j_k \times p_k}$, *and* $\boldsymbol{b}_k \in \mathbb{R}^{p_k}$. *For each layer* $k=\{1,2,\ldots,m\}$, *we describe the function* h_k *as in* (V.3).

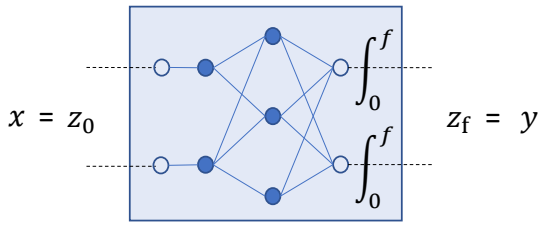


Figure V.1: Illustration of an example of NODE, as defined in (V.1) and (V.4), and Definition 3.

An example of NODE with m = 2 hidden layers, 2 inputs and 2 outputs is depicted in Figure V.1.

V.2.1 General Neural ODE

The general class of neural ODEs (GNODEs) considered in this work is more complex than previously analyzed neural ODEs as it may be comprised of two types of layer: NODEs and $\mathcal{NN}$-Layers. We introduce a more general framework where multiple NODEs can make up part of the overall architecture along with other $\mathcal{NN}$-Layers, as described in Definition 4. This is the reachability problem subject of evaluation in this work.

Definition 4 (GNODE) *A* ***GNODE*** $\mathscr{F}$ *is any sequence of consecutively connected* N *layers* $\mathscr{L}_k$ *for* $k \in \{1,\ldots N\}$ *with* N_O *NODEs and* N_D $\mathcal{NN}$*-Layers, that meets the conditions* $1 \leq N_O \leq N$, $0 \leq N_D < N$, *and* $N_D + N_O = N$.

With the above definition, we formulate a theorem for the restricted class of neural ODEs (NODE) that we use to compare our methods against existing techniques.

Observation 1 (Special case 1: NODE) *Let* $\mathscr{F}$ *be a GNODE with* N *layers. If* $N = 1$, $N_O = 1$ *and* $N_D = 0$, *then* $\mathscr{F}$ *is equivalent to an ODE whose continuous-time dynamics are defined as a neural network, and we refer to it as a* ***NODE*** *(as in Definition 3).*

In Figure V.2, an example of a GNODE is shown, which has 1 input (x), 1 output (y), N_O = 2 NODEs, and N_D = 5 $\mathcal{NN}$-Layers, with its 5 numbered segments described as: 1) The first segment has a $\mathcal{NN}$-Layer, with one hidden layer of 2 neurons and an output layer of 2 neurons, 2) a NODE with one hidden layer of 3 neurons and an output layer of 2 neurons, 3) $\mathcal{NN}$-Layer with 3 hidden layers of 4,1, and 2 neurons respectively, and an output layer of 2 neurons, 4) NODE with a hidden layer of 3 neurons and output layer of 2 neurons, and 5) $\mathcal{NN}$-Layer with a hidden layer of 4 neurons and and output layer with 1 neurons.

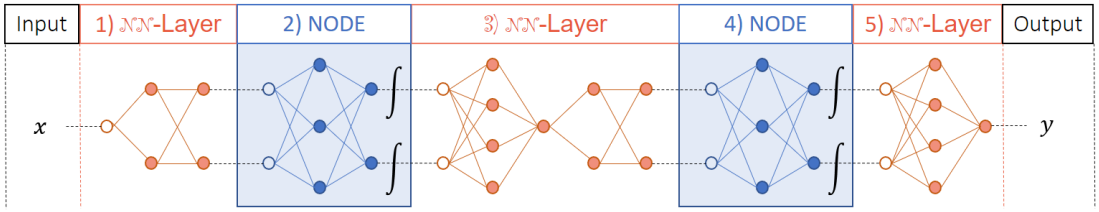


Figure V.2: Example of a GNODE. The filled circles represent weighted neurons in each layer, non-filled neurons represent the inputs of each layer-type segment, shown for visualization purposes.

Given this GNODE architecture, we are capable of encoding the originally proposed neural ODE [6], as well as several of its model improvements including higher-order neural ODEs like the second-order neural ODE (SONODE) [87], augmented neural ODEs (ANODEs) [84], and input-layer augmented neural ODEs (ILNODEs). This general class of neural ODEs from Definition 4 is applicable to many applications including image classification, time series prediction, dynamical system modeling and normalizing flows.

Observation 2 (Special case 2 - NNCS) *We can consider the NNCS as a special case of the* GNODE*, as the NNCS models also consist of* $\mathcal{NN}$*-Layers and NODE, under the assumption that the plant is described as an NODE. NNCS in the general architecture have* $\mathcal{NN}$*-Layers followed by an ODE or NODE, which connects back to the* $\mathcal{NN}$*-Layers in a feedback loop manner for* cp *control steps. Thus, by unrolling the NNCS* cp *times, we consecutively connect the* $\mathcal{NN}$*-Layers with the NODE, creating a* GNODE*.*

V.2.2 NODE: Applications

There are two application modes of a NODE, model reduction and dynamical systems. On one hand, we can use it as a substitute for multiple similar layers to reduce the depth and overall size of a model [6]. In this context, we treat the NODE as an input-output mapping function, and set the integration time t_f to a fixed number during training, which will be then used during simulation or reachability. Typically, this value is set to 1. On the other hand, we can make use of NODEs to capture the behavior of time-series data or dynamical systems. In this sense, t_f is not fixed, and it will be determined by the user/application. In summary, we can use NODEs for: 1) time-series or dynamical system modeling, and 2) model reduction. For time-series, t_f is variable, user or application dependent. For model reduction, t_f is a parameter of the model fixed before training. Only the output value at time = t_f is used, while for the time-series models, we are interested in the interval [0,t_f].

V.2.3 Reachability Analysis

The main focus of this manuscript is to introduce a framework for the reachability analysis of GNODEs. This framework combines set representations and methods from Neural Network (NN), Convolutional Neural

Network (CNN) and hybrid systems reachability analysis. Similar to neural network reachability in NNV [14], we consider the set propagation through each layer as well as the set conversion between layers for all reachability methods for the supported layers. Hence, the reachability problem of GNODE is defined as follows:

Definition 5 (Reachable Set of a GNODE) *Let $\mathscr{F}$: $\mathbb{R}^j \to \mathbb{R}^p$ be a **GNODE** with N layers. The output reachable set $\mathscr{R}_N$ of a **GNODE** with input set $\mathscr{R}_0$ is defined as:*

$$\begin{aligned}
\mathscr{R}_1 &\triangleq \{y_1 \mid y_1 = f_1(y_0),\ y_0 \in \mathscr{R}_0\},\\
\mathscr{R}_2 &\triangleq \{y_2 \mid y_2 = f_2(y_1),\ y_1 \in \mathscr{R}_1\},\\
&\vdots\\
\mathscr{R}_N &\triangleq \{y_N \mid y_N = f_N(y_{N-1}),\ y_{N-1} \in \mathscr{R}_{N-1}\},
\end{aligned} \tag{V.5}$$

where f_i: $\mathbb{R}^{j_i} \to \mathbb{R}^{p_i}$ is the function f of layer i, where f is either a NODE (g) or a $\mathcal{NN}$-Layer (h).

The proposed solution to this problem is sound and incomplete, in other words, an over-approximation of the reachable set. This means that, given a set of inputs, we compute an output set of which, given any point in the input set, we simulate the GNODE and the output point is always contained within the reachable output set (sound). However, it is incomplete because the opposite is not true; there may be points in the output reachable set that cannot be traced back to be within the bounds of the input set.

Definition 6 (Soundness) *Let $\mathscr{F}$: $\mathbb{R}^j \to \mathbb{R}^p$ be a GNODE with an input set $\mathscr{R}_0$ and output reachable set $\mathscr{R}_f$. The computed $\mathscr{R}_f$ given $\mathscr{F}$ and $\mathscr{R}_0$ is **sound** iff $\forall x \in \mathscr{R}_0$, $\mid y = \mathscr{F}(x)$, $y \in \mathscr{R}_f$.*

Definition 7 (Completeness) *Let $\mathscr{F}$: $\mathbb{R}^j \to \mathbb{R}^p$ be a GNODE with an input set $\mathscr{R}_0$ and output reachable set $\mathscr{R}_f$. The computed $\mathscr{R}_f$ given $\mathscr{F}$ and $\mathscr{R}_0$ is **complete** iff $\forall x \in \mathscr{R}_0$, $\exists y = \mathscr{F}(x) \mid y \in \mathscr{R}_f$ and $\forall y \in \mathscr{R}_f$, $\exists x \in \mathscr{R}_0$ $\mid y = \mathscr{F}(x)$.*

Definition 8 (Reachable set of a $\mathcal{NN}$-Layer) *Let h: $\mathbb{R}^j \to \mathbb{R}^p$ be a $\mathcal{NN}-$Layer as described in Definition 2. The reachable set $\mathscr{R}_h$, with input $\mathscr{X} \subset \mathbb{R}^n$ is defined as*

$$\mathscr{R}_h = \{y \mid y = h(x), x \in \mathscr{X}\}.$$

Definition 8 applies to any discrete-time layer that is part of a GNODE, including, but not limited to the following supported $\mathcal{NN}-$*Layers* supported in NNVODE: fully-connected layers with ReLU, tanh, sigmoid and leaky-ReLU activation functions, and convolutional-type layers such as batch normalization, 2-D convolutional, and max-pooling.

The reachability analysis of NODEs is akin to the general reachability problem for any continuous-time system modeled by an ODE.

If we represent the NODE as a single layer i of a GNODE with continuous dynamics described by (V.1), and assume that for a given initial state $z_0 \in \mathbb{R}^{j_i}$, the system admits a unique trajectory defined on $\mathbb{R}_0^+$, described by $\zeta(.,z(t_0))$, then the reachable set of the given NODE can be characterized by Definition 9.

Definition 9 (Reachable set of a NODE) *Let g be a **NODE** with solution $\zeta(t; z_0)$ to* (V.1) *for initial state z_0. The reachable set, $\mathscr{R}_g$ at $t = t_F$, $\mathscr{R}_g(t_F)$, with initial set $\mathscr{R}_0 \subset \mathbb{R}^n$ at time $t = t_0$ is defined as*

$$\mathscr{R}_g(t_F) = \{\zeta(t;z_0) \in \mathbb{R}^n \mid z_0 \in \mathscr{R}_0, t \in [0,t_f]\}.$$

We also describe the reachable set for a time interval $[t_0,t_f]$ as follows

$$\mathscr{R}_g([t_0,t_f]) := \bigcup_{t\in[t_0,t_f]} \mathscr{R}_g(t)$$

Now that we have outlined the general reachability problem of a NODE, whether we compute a single reachable set for the NODE at $t = t_F$, or over an interval $t \in [t_0,t_F]$ (computed by *get_time()*), depends on how the neural ODE was trained and the specific application of its use. However, the core computation remains the same. This is outlined in Algorithm 4.

Algorithm 4: Reachability analysis of a GNODE.

Result: $\mathscr{R}_f$ //output reachable set
Input: $\mathscr{F}, \mathscr{R}_0$ // GNODE, input set
N = $\mathscr{F}$.layers // number of layers in GNODE
for i = *1 : N* **do**
  $\mathcal{L}_i$ = $\mathscr{F}$.layer(i)
  if $\mathcal{L}_i$ *is NODE* **then**
    $\mathbf{t}_i$ = get_time($\mathcal{L}_i$) // get integration time bounds of layer i
    $\mathscr{R}_i$ = reach$_{NODE}$($\mathcal{L}_i$,$\mathscr{R}_{i-1}$, $\mathbf{t}_i$) // reach set of layer i, Definition 9
  else
    $\mathscr{R}_i$ = reach$_{\mathcal{NN}}$($\mathcal{L}_i$, $\mathscr{R}_{i-1}$) // reach set of layer i, Definition 8
  end
end
$\mathscr{R}_f = \mathscr{R}_N$ // output reachable set
return: $\mathscr{R}_f$

V.2.4 Reachability Methods

In the previous sections, we defined the reachable set of a GNODE using a layer-by-layer approach. However, the computation of these reachable sets is defined by the reachability methods and set representations utilized

in their construction. For instance, the same operations are not utilized to compute the reachable set for a fully-connected layer with a hyperbolic tangent activation function as with a fully convolutional layer. In the following section, we describe the set of methods in the NNVODE tool available to the community to compute the reachable set of each specific layer.

We begin with the NODE approaches, where we make a distinction based on the underlying dynamics. If the NODE is nonlinear, we make use of zonotope and polynomial-zonotope based methods that are implemented and available in CORA [65, 150]. If the NODE is purely linear, then we utilize the star set-based methods introduced in [72], which are more scalable than other zonotope-based methods and possess soundness guarantees as well.

For the $\mathcal{NN}$-Layers, there are several methods available, including zonotope-based and star-set based methods. However, we limited our implementation only using star sets to handle these layers, as this representation was demonstrated to be more computationally efficient and to enable tighter over-approximations of the derived reachable sets than zonotope methods [108]. Additionally, in using star set methods, we allow for the use of both approximate methods (*approx-star*) and exact methods (*exact-star*). A summary of these methods and supported layers is depicted in Table V.1, and for a complete description of the reachability methods utilized in our work, we refer the reader to [14] and the manual[1][2].

Table V.1: Layers supported in NNVODE and reachability sets and methods available.

GNODE part	Layer Type – Set Representation (method name)
NODE	Linear – Star-set (”direct”) [72]
NODE	Nonlinear – Zonotope, Polynomial Zonotope * [65, 150]
$\mathcal{NN}$-Layer	FC: linear, ReLU – Star-set (”approx-star”, ”exact-star”) [108]
$\mathcal{NN}$-Layer	FC: leakyReLU, tanh, sigmoid, satlin – Star-set (”approx-star”) [14]
$\mathcal{NN}$-Layer	Conv2D – ImageStar (”approx-star”, ”exact-star”) [14]
$\mathcal{NN}$-Layer	BatchNorm – ImageStar (”approx-star”, ”exact-star”) [14]
$\mathcal{NN}$-Layer	MaxPooling2D – ImageStar (”approx-star”, ”exact-star”) [14]
$\mathcal{NN}$-Layer	AvgPooling2D – ImageStar (”approx-star”, ”exact-star”) [14]

* We support several methods available using Zonotope and PolyZonotopes, which includes user-defined fixed reachability parameters (”ZonoF”, ”PolyF”) as well as adaptive reachability methods (”ZonoA” and ”PolyA”), which require no prior knowledge on reach methods or systems to verify to produce relevant results.

One of the key aspects of the verification of GNODEs is the proper encoding of the NODEs within reachability schemes. Depending on the software that is used, this process may vary and require distinct steps. As an example, some tools, like NNVODE, are simpler and allow for matrix multiplications within the definition of equations. However, for tools like Flow*, the set of steps required to properly encode this problem are more complex as it requires a definition for each individual equation of the state derivative. Thus, a more general conversion is needed, which is illustrated in Section V.2.5.

[1] NNV manual is available at: https://github.com/verivital/nnv/blob/master/docs/manual.pdf
[2] CORA manual (Release 2021) is available at: https://tumcps.github.io/CORA/data/Cora2021Manual.pdf

V.2.5 Implementation Example of a NODE

Let the function $g(z)$ be composed of two fully-connected layers, where σ_1 and σ_2 are *tanh* and *linear* activation functions, respectively. The state, $z \in \mathbb{R}^2$, is a two-dimensional vector, the first layer contains five neurons and the second layer has two neurons. Therefore, the parameters of the layers are $W_1 \in \mathbb{R}^{5\times 2}$, $\mathbf{b}_1 \in \mathbb{R}^5$, $W_2 \in \mathbb{R}^{2\times 5}$ and $\mathbf{b}_2 \in \mathbb{R}^2$. We can define the state derivatives of this NODE as:

$$\dot{z} = g(z) = L_2(L_1(z)) = \sigma_2(W_2 \times (\sigma_1(W_1 \times z + \mathbf{b}_1) + \mathbf{b}_2) = W_2 \times (tanh(W_1 \times z + \mathbf{b}_1) + \mathbf{b}_2. \tag{V.6}$$

For NNVODE as well as JuliaReach or GoTube, the representation in (V.6) is a valid format. However, for consistency and comparison purposes, other tools like Flow* require the explicit equation for every state z_i. Let the states z, weights W_k and biases $\mathbf{b}_k$ be

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, W_1 = \begin{bmatrix} w_{1_{11}} & w_{1_{12}} \\ w_{1_{21}} & w_{1_{22}} \\ w_{1_{31}} & w_{1_{32}} \\ w_{1_{41}} & w_{1_{42}} \\ w_{1_{51}} & w_{1_{52}} \end{bmatrix}, \mathbf{b}_1 = \begin{bmatrix} b_{1_1} \\ b_{1_2} \\ b_{1_3} \\ b_{1_4} \\ b_{1_5} \end{bmatrix}, W_2 = \begin{bmatrix} w_{2_{11}} & w_{2_{12}} & w_{2_{13}} & w_{1_{14}} & w_{2_{15}} \\ w_{2_{21}} & w_{2_{22}} & w_{2_{23}} & w_{1_{24}} & w_{2_{25}} \end{bmatrix}, \mathbf{b}_2 = \begin{bmatrix} b_{2_1} \\ b_{2_2} \end{bmatrix}, \tag{V.7}$$

then we can explicitly write out the full equations for every state derivative:

$$\begin{aligned}
\dot{z}_1 &= w_{2_{11}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{12}} tanh(w_{1_{21}} z_1 + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{13}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\
&\quad + w_{1_{14}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) + w_{2_{15}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1}, \\
\dot{z}_2 &= w_{2_{21}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{22}} tanh(w_{1_{21}} z_1 + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{23}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\
&\quad + w_{1_{24}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) + w_{2_{25}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1},
\end{aligned} \tag{V.8}$$

For the special case where σ_k is linear for every layer $k \in m$, we can simplify the NODE as a linear ODE:

$$\begin{aligned}
\dot{z} &= (W_m W_{m-1} \dots W_1) z + \mathbf{b}_m + \mathbf{b}_{m-1} W_m + \mathbf{b}_{m-2} W_m W_{m-1} + \mathbf{b}_1 W_m \dots W_2 \\
\dot{z} &= (W_m W_{m-1} \dots W_1) z + \mathbf{b}_1 W_m \dots W_2 + \mathbf{b}_2 W_m \dots W_3 + \mathbf{b}_{m-1} W_m + \mathbf{b}_m \\
\dot{z} &= \prod_{i=1}^{m} (W_i) z + \sum_{i=1}^{m-1} (\mathbf{b}_i \prod_{j=i+1}^{m} (W_j)) + \mathbf{b}_m, \\
\dot{z} &= Az + Bu + c,
\end{aligned} \tag{V.9}$$

where A is equal to weights term, $A = \prod_{i=1}^{m}(W_i)$, $B = 0$, and the rest of the terms of the equation corresponds to the constant vector

$c = \sum_{i=1}^{m-1} (\mathbf{b}_i \prod_{j=i+1}^{m}(W_j)) + \mathbf{b}_m$.

V.3 Evaluation

Having described the details of our reachability definitions, algorithm, and implementation, we now present the experimental evaluation of our proposed work. We begin by presenting, a method and tool comparison analysis against GoTube[3] [104], Flow*[4] [66] and JuliaReach[5] [71]. Then, we present a case study of an Adaptive Cruise Control system, and conclude with an evaluation of the scalability of our techniques using a random set of architectures for dynamical system applications as well as a set of classification models for MNIST. The GNODE architectures for each benchmark can be found in the Appendix V.6. To facilitate the reproducibility of our experiments, we set a timeout of 2 hours (7200 seconds) for each reach set computation[6]. All our experiments were conducted on a desktop with the following configuration: Intel Core i7-7700 CPU @ 3.6GHz 8 core Processor, 64 GB Memory, and 64-bit Ubuntu 16.04.3 LTS OS.

V.3.1 Method and Tool Comparison

We have implemented several methods within NNVODE, and the first evaluation consists of comparing the available methods for nonlinear NODEs, fixed-step zonotope and polynomial zonotopes (zono-F,poly-F) and adaptive zonotope and polynomial zonotope (zono-A,poly-A) based methods [65]. For all other $\mathcal{NN}$-Layers in the GNODEs, we use the star-set over-approximate methods. We considered multiple models, all inspired by the ILNODE representation that was introduced by Massaroli. They consist of a set of models with a varying number of augmented dimensions. All these models are instances of the GNODE class presented in Definition 4, which present an architecture of the form $\mathcal{NN}$-Layers + NODE + $\mathcal{NN}$-Layers. In this context, we were concerned with how the methods scale with respect to the number of dimensions of each model.

Table V.2: Computation time of the reachability analysis of the Damped Oscillator benchmark. Results are shown in seconds with up to one decimal place.

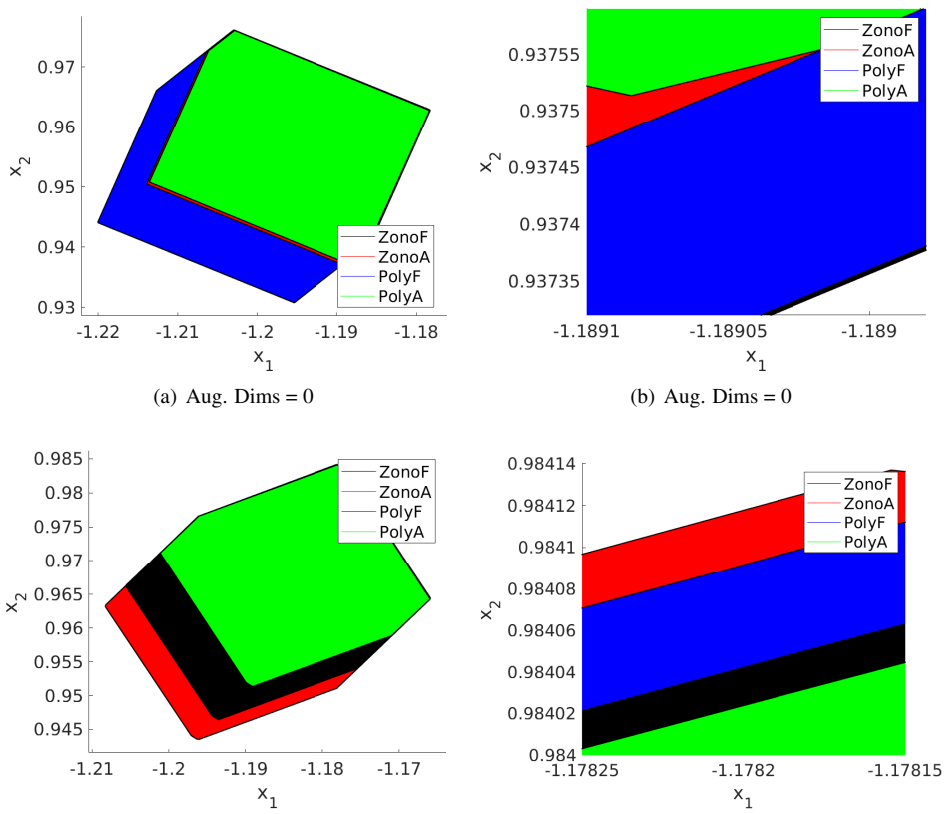
Aug. Dims	Zono-*F*	Zono-*A*	Poly-*F*	Poly-*A*
0	**34.0**	574.5	201.7	654.0
1	**146.4**	4205.0	1573.9	3440.9
2	**441.0**	–	–	–

[3] GoTube can be found at https://github.com/DatenVorsprung/GoTube

[4] Flowstar version 2.1.0 is available at https://flowstar.org/

[5] JuliaReach can be found at https://juliareach.github.io/

[6] Code to reproduce all results can be found here: https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/FORMATS2022



(a) Aug. Dims = 0 (b) Aug. Dims = 0

(c) Aug. Dims = 1 (d) Aug. Dims = 1

Figure V.3: Reach sets comparison of the Damped Oscillator benchmark of the model with 0 and 1 augmented dimensions. Plots (a) and (c) show the reachable set at t = 1s, and plots b) and d) show the zoomed-in reach sets to observe the minor size differences between the four methods: **ZonoA**, **ZonoF**, **PolyF** and **PolyA**.

In Table V.2, we observe that the Zono-*F* method is the fastest across all models, while the adaptive methods are the slowest. Moreover, only Zono-*F* is able to complete the reach set computation for all three models, while the other methods time out. In terms of the size of the computed reach sets, we compare the last reach set obtained in Figure V.3. In the subplots V.3(b) and V.3(d) we see the zoomed in reach sets, and observe that the Poly-*A* method computes the smallest over-approximate reach set across both experiments. Based on these results, in all subsequent experiments, we use the *zono-F* method for nonlinear NODEs, the *direct* method for linear NODEs, and the over-approximate star-set methods for all the $\mathcal{NN}$-Layers.

The next part of our evaluation consists of comparing NNVODE's methods for NODEs to those of Flow* [66], GoTube [104] and JuliaReach [71] across a collection of benchmarks that include a linear and non-linear 2-dimensional spiral [6], a Fixed-Point Attractor (FPA) [151] and a controlled cartpole [103]. The computation reachability results are displayed in Table V.3 with the intention to characterize major differences between tools, i.e., to show some tools are 10× to 20× faster than others for some benchmarks. It is

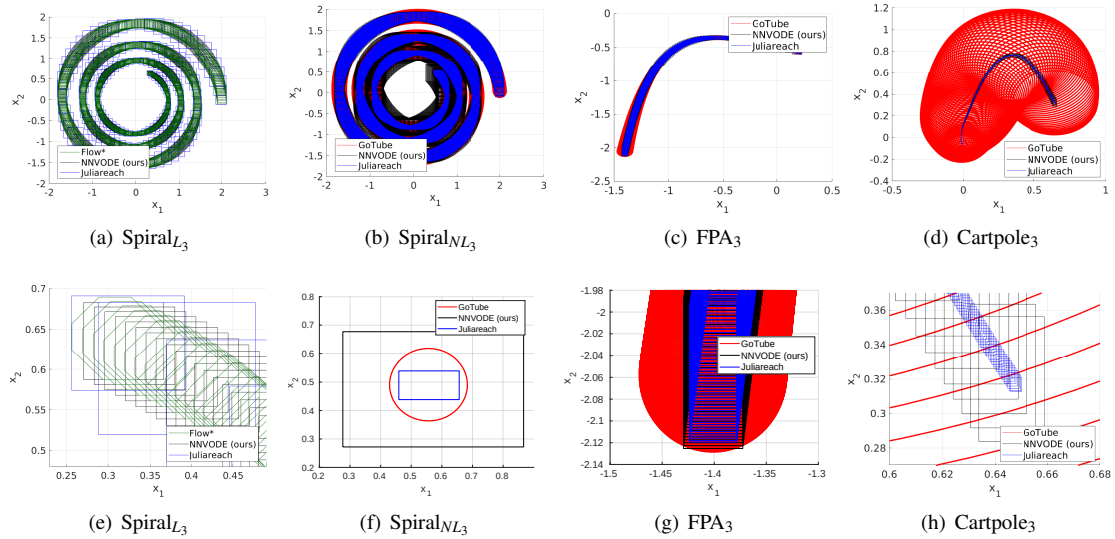
worth noting, that we are not experts on every tool that we considered. Thus, it may be possible to optimize the reachable set computation for each benchmark with depending on the tool. However, we did not do so. Instead, we attempted around 3 to 5 different parameter combinations for each benchmark, and used the best results we could obtain when comparing against the other tools. The details can be found in the Appendix V.7. The first three rows correspond to the linear spiral 2D model, and the subsequent three rows to the nonlinear one. The first set of observations that can be made in this context is that Flow* times out on all problems involving nonlinear neural ODEs, and that GoTube cannot obtain a solution to the reachability problem for the linear model. In terms of computation time, the results vary. Flow* is the fastest for the linear Spiral 2D model, regardless of the size of the initial set. In general, JuliaReach and NNVODE are much faster than GoTube, with JuliaReach being the fastest tool across the board. Additionally, there is not a significant difference between NNVODE and JuliaReach in the treatment of the nonlinear spiral model and FPA models. However, JuliaReach is an order of magnitude faster than NNVODE and two orders of magnitude faster than GoTube on the cartpole benchmark. In Figure V.4 we display a subset of the reachability results from Table V.3 where we observe that JuliaReach is able to compute smaller over-approximations of the reachable set on all the benchmarks except for the linear spiral model. Notably, in most cases, GoTube computes the largest over-approximation, and this effect grows far more significantly than the other tools when the complexity of the model increases. This can be observed in Figures V.4(d) and V.4(h).

Table V.3: Results of the reachability analysis of the NODE benchmarks. All results are shown in seconds. TH stands for Time Horizon and δ_μ to the input uncertainty.

Name	TH (s)	δ_μ	Flow*	GoTube	JuliaReach	NNVODE (ours)
Spiral_{L_1}	10	0.01	**4.0**	–	10.2	8.0
Spiral_{L_2}	10	0.05	**3.7**	–	7.2	7.4
Spiral_{L_3}	10	0.1	**3.7**	–	7.2	7.3
Spiral_{NL_1}	10	0.01	–	106.4	58.9	**47.4**
Spiral_{NL_2}	10	0.05	–	106.7	**41.7**	46.2
Spiral_{NL_3}	10	0.1	–	106.5	**41.9**	46.2
FPA_1	0.5	0.01	-	13.6	1.9	**1.3**
FPA_2	2.5	0.01	-	42.4	**5.6**	6.6
FPA_3	10.0	0.01	-	140.2	28.4	**7.5**
Cartpole_1	0.1	$1e$-4	-	183.0	**3.2**	67.9
Cartpole_2	1.0	$1e$-4	-	1590.9	**13.8**	404.3
Cartpole_2	2.0	$1e$-4	-	3065.7	**35.5**	834.7

V.3.2 Case Study: Adaptive Cruise Control (ACC)

This case study was selected to evaluate an original NNCS benchmark used in all the AINNCS ARCH-Competitions [13, 76, 77, 14] against NNCS with NODEs as dynamical plants learned from simulation data using 3^{rd} order NODEs [87]. We demonstrate the verification of the ACC with different GNODEs learned as

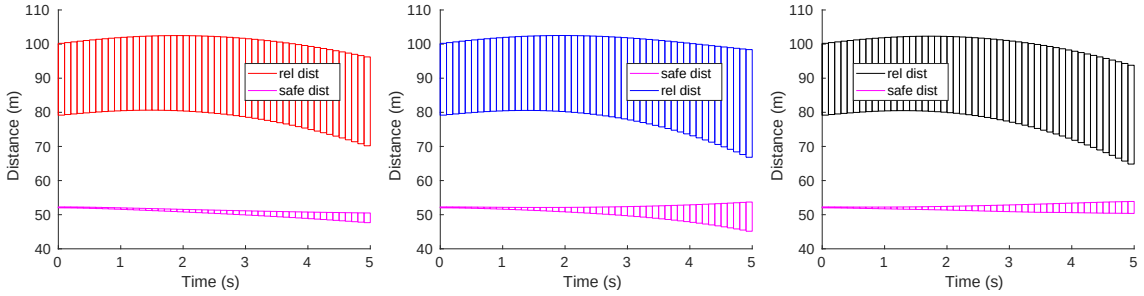


(a) Spiral$_{L_3}$ (b) Spiral$_{NL_3}$ (c) FPA$_3$ (d) Cartpole$_3$

(e) Spiral$_{L_3}$ (f) Spiral$_{NL_3}$ (g) FPA$_3$ (h) Cartpole$_3$

Figure V.4: NODE reach set comparisons. The top 4 figures ([*a*,*d*]) show the complete reachable sets of each benchmark, while the bottom 4 correspond to the zoomed-in reach sets (*e*, *g*) and to the zoomed-in figure of the last reach set (*f*,*h*). The figures show the computed reach sets of **GoTube**, **NNVODE**, **JuliaReach** and **Flow***.

the plant model of the ACC and compare against the original benchmark, while using the same NN controller across all three models. The details of the original ACC NNCS benchmark can be found in [14], and the architectures of the third order neural ODEs can be found in the Appendix V.6.



(a) Original (b) Linear NODE (c) Nonlinear NODE

Figure V.5: Adaptive Cruise Control comparison. In red we display the relative distance of the original plant, in blue the linear NODE, in **black** the nonlinear NODE, and in magenta, the safe distance.

We considered all three models using the same initial conditions and present the results in Figure V.5. The reachable sets obtained from all three plants are largely similar, and we can guarantee that all the models are safe since the intersection between the safe distance and the relative distance is empty. When we consider the size of the reachable sets, one can see that the original model returns the smallest reach set, whereas the linear model boasts the largest one. In all, the biggest difference between the plant dynamics is the computation

time. Here, the linear 3^{rd} order NODE boasts the fastest computation time of 0.86 seconds, whereas the original plant takes 14.9 seconds, and the nonlinear 3^{rd} order NODE takes 998.7 seconds.

V.3.3 Classification NODEs

Our second set of experiments considered performing a robustness analysis for a set of MNIST classification models with only fully-connected $\mathcal{NN}$-Layers and other 3 models with convolutional layers as well. There is one linear NODE in each model, and we vary the number of parameters and states across all of them to study the scalability of our methods. We evaluate the robustness of these models under an L_∞ adversarial perturbation of $\epsilon = \{0.05, 1, 2\}$ over all the pixels, and $\epsilon = \{2.55, 12.75, 25.5\}$ over a subset of pixels (80). [7] The complete evaluation of this benchmark consists of a robustness analysis using 50 randomly sampled images for each attack. We compare the number of images the neural ODEs are robust to, as well as the total computation time to run the reachability analysis for each model in Table V.4.

Definition 10 (Robustness) *Given a classification-based GNODE $\mathscr{F}(z)$, input $z \in \mathbb{R}^j$, perturbation parameter $\epsilon \in \mathbb{R}$ and an input set Z_p containing z_p such that $Z_p = \{z : ||z - z_p|| \leq \epsilon\}$ that represents the set of all possible perturbations of z. The neural ODE is locally **robust** at z if it classifies all the perturbed inputs z_p to the same label as z, i.e., the system is **robust** if $\mathscr{F}(z_p) = \mathscr{F}(z)$ for all $z_p \in Z_p$.*

Table V.4: Robustness analysis of MNIST classification GNODEs under L_∞ adversarial perturbations. The accuracy and robustness results are described as percentage values between 0 and 1, and the time computation corresponds to the average time to compute the reachable set per image. Columns 3-8 corresponds to $\epsilon = \{0.5,1,2\}$ over all pixels in the image, columns 9-14 corresponds to $\epsilon = \{2.55,12.75,25.5\}$ attack over a subset of pixels (80) n each image.

		ℓ_∞						$\ell_{\infty\ (80)}$					
		0.5		**1**		**2**		**2.55**		**12.75**		**25.5**	
Name	**Acc.**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**
$FNODE_S$	0.9695	1	0.0836	0.98	0.1062	0.98	0.1186	1	0.0192	0.98	0.0193	0.98	0.0295
$FNODE_M$	0.9772	1	0.0948	1	0.1177	0.98	0.2111	1	0.0189	1	0.0225	0.98	0.0396
$FNODE_L$	0.9757	1	0.1078	1	0.1674	0.96	0.5049	1	0.0187	1	0.0314	0.96	0.0709
$CNODE_S$	0.9706	0.98	13.95	0.96	15.82	0.86	17.58	0.98	1.943	0.94	3.121	0.78	4.459
$CNODE_M$	0.9811	1	176.5	1	193.8	1	293.8	1	21.45	1	83.28	1	182.0
$CNODE_L$	0.9602	1	1064	1	1089	1	1736	1	234.6	1	522.7	1	779.3

Finally, we performed a scalability study using a set of random GNODE architectures with multiple NODEs. Here, we focus on the nonlinear methods for both the $\mathcal{NN}$-Layers and the NODEs and evaluate how our methods scale with the number of neurons, inputs, outputs, and dimensions in the NODE. The main challenge of these benchmarks is the presence of multiple NODEs within the GNODE. There are a total of 6 GNODEs (XS,S,M,L,XL, and XXL), all with the same number of layers. However, we increase the number

[7] Adversarial perturbations are applied before normalization, pixel values $z_p \in [0,255]$.

of inputs, outputs, and parameters across all the layers of the GNODEs. Here, XS corresponds to the smallest model and XXL to the largest. A description of these architectures can be found in the Appendix V.6.

Several trends can be observed from Table V.5. The first is that in general, smaller models have smaller reach set computation times, with two notable exceptions: the first run with XS and the last experiment with XL. Furthermore, one can observe that the largest difference in the reach set computation times comes from increasing the number of states in the NODEs, which are $\{2,3,4,4,5,5\}$ for both NODEs in every model in $\{$XS,S,M,L,XL, and XXL$\}$ respectively, while increasing the input and output dimensions ($\{1,2,2,3,3,4\}$ respectively) does not affect the reachability computation as much. This is because of the complexity of nonlinear ODE reachability as state dimensions increase, while for $\mathcal{NN}-$*Layers*, increasing the size of the inputs or neurons by 1 or a few units does not affect the reachability computation as much.

Table V.5: Computation time of the reachability analysis of the randomly generated GNODEs. Results are shown in seconds.

	XS	S	M	L	XL	XXL
$\delta_\mu = 0.01$	57.0	16.2	59.3	61.8	168.3	223.9
$\delta_\mu = 0.02$	3.4	11.4	42.2	41.0	262.4	115.4
$\delta_\mu = 0.04$	3.1	10.3	37.6	72.9	1226.3	243.6

V.4 Related Work

Analysis of Neural ODEs. To the best of our knowledge, this is the first empirical study of the formal verification of neural ODEs as presented in the general neural ODE (GNODE) class. Some other works have analyzed neural ODEs, but are limited to a more restricted class of neural ODEs with only purely continuous-time models. We refer to these models in this chapter as NODEs. The most comparable work is a theoretical inquiry of the neural ODE verification problem using Stochastic Lagrangian Reachability (SLR) [102], which was later extended and implemented in a stochastic reachability analysis tool called GoTube [104]. The SLR method is an abstraction-based technique that is able to compute a tight over-approximation of the set of reachable states, and provide stochastic guarantees in the form of confidence intervals for the derived reachable set. Beyond reachability analysis, there have also been several works investigating the robustness of neural ODEs. In [96], the robustness of neural ODE image classifiers is empirically evaluated against residual networks, which are a standard deep learning model for image classification tasks. Their analysis demonstrates that neural ODEs are more robust to input perturbations than residual networks. In a similar work, a robustness comparison of neural ODEs and standard CNNs was performed [97]. Their work considers two standard adversarial attacks, Gaussian noise and FGSM [3], and their analysis illustrate that neural ODEs are more robust to adversarial perturbations. In terms of the analysis of GNODEs, to the best of

our knowledge, no comparable work as been done, although for specific models where all the $\mathcal{NN}$-Layers of a GNODE have fully-connected layers with continuous differentiable activation functions like sigmoid or tanh, it may be possible to compare our methods to other tools (with minor modifications) like Verisig [50, 51] or JuliaReach [71]. However, that would restrict the more general class of neural ODEs (GNODEs) that we evaluate in this manuscript.

Table V.6: Summary of related verification tools. A ✓means that the tool supports verification of this class, a ◯ means that it may be supported it, but some minor changes may be needed, a – means it does not support it, and a ⊙ means that some small changes have been made to the tool for comparison and the tools has been used to verify at least one example on this class.

Tool	ODE[8]	NODE[9]	NN[10]	NNCS	GNODE
CORA [65]	✓	⊙	–	–	–
ERAN [64]	–	–	✓	–	–
Flow* [66]	✓	⊙	–	–	–
GoTube [104]	✓	✓	–	–	–
JuliaReach [71]	✓	⊙	✓	✓	–
Marabou [148]	–	–	✓	–	–
nnenum [147]	–	–	✓	–	–
ReachNN [70, 58]	✓	◯	✓	✓	–
Reluplex [130]	–	–	✓	–	–
ReluVal [83]	–	–	✓	–	–
Sherlock [49]	✓	◯	✓	✓	–
SpaceEx [67]	✓	◯	–	–	–
Verisig [50, 51]	✓	◯	✓	✓	–
NNV [14]	✓	⊙	✓	✓	–
NNVODE (ours)	✓	✓	✓	✓	✓

[8]ODE verification is considered to be supported for any tool that can verify at least one of linear or nonlinear continuous-time ODEs. [9]NODE is a specific type of ODE, so in theory any tool that supports ODE verification may be able to support NODE. However, these tools are optimized for NODE and in practice may not be able to verify most of these models as seen on our comparison with Flow*. [10]We include in this category any tool that supports one or more verification methods for fully-connected, convolutional or pooling layers.

Verification of Neural Networks and Dynamical systems. The considered GNODEs are a combination of dynamical systems equations (ODEs) modeled using neural networks, and neural networks. When these subjects are treated in isolation, one finds that there are numerous studies that consider the verification of dynamical systems, and correspondingly there are numerous works that deal with the neural network verification problem. With respect to the former, the hybrid systems community has considered the verification of dynamical systems for decades and developed tools such as SpaceEx [67], Flow* [66], CORA [65], and JuliaReach [71] that deal with the reachability problem for discrete-time, continuous-time or even hybrid dynamics. A more comprehensive list of tools can be found in the following paper [152]. Within the realm of neural networks, the last several years have witnessed numerous promising verification methods proposed towards reasoning about the correctness of their behavior. Some representative tools include Reluplex [130], Marabou [148], ReluVal [83], NNV [14] and ERAN [64]. The tools have drawn inspiration from a wide range of techniques including optimization, search, and reachability analysis [60]. A discussion of these ap-

proaches, along with pedagogical implementations of existing methods, can be found in the following paper [60]. Building on the advancements of these fields, as a natural progression, frameworks that consider the reachability problem involving neural networks and dynamical systems have also emerged. Problems within the space are typically referred to as Neural Network Control Systems (NNCS), and some representative tools include NNV [14], Verisig [50, 51], and ReachNN* [70, 58]. These tools have demonstrated their capabilities and efficiency in several works including the verification of an adaptive cruise control (ACC) [14], an automated emergency breaking system [55], and autonomous racing cars [153, 154] among others, as well as participated in the yearly challenges of NNCS verification [77, 76, 13]. These studies and their respective frameworks are very closely related to the work contained herein. In a sense, they deal with a restricted GNODE architecture, since their analysis combines the basic operations of neural network and dynamical system reachability in a feedback-loop manner. However, this restricted architecture would only consist of a fully-connected neural network followed by a NODE. In Table V.6, we present a **comparison of verification tools** that can support one or more of the following verification problems: the analysis of continuous-time models with linear, nonlinear or hybrid dynamics (ODE), neural networks (NN), neural network control systems (NNCS), neural ODEs (NODE) and GNODEs as presented in Figure V.2.

V.5 Conclusion & Future Work.

We have presented a verification framework to compute the reachable sets of a general class of neural ODEs (GNODE) that no other existing methods are able to solve. We have demonstrated through a comprehensive set of experiments the capabilities of our methods on dynamical systems, control systems and classification tasks, as well as the comparison to state-of-the-art reachability analysis tools for continuous-time dynamical systems (NODE). One of the main challenges we faced was the scalability of the nonlinear ODE reachability analysis as the dimension complexity of the models increased, as observed in the cartpole and damped oscillator examples. Possible improvements include integrating other methods into our framework such as [155], which improves the current nonlinear reachability analysis via an improved hybridization technique that reduces the sizes of the linearization domains, and therefore reduces overapproximation error. Another approach would be to make use of the Koopman Operator linearization prior to analysis in order to compute the reach sets of the linear system, which are easier and faster to compute as observed from the experiments conducted using the two-dimensional spiral benchmark [156]. In terms of models and architectures, latent neural ODEs [12] and some of its proposed variations such as controlled neural DEs [93] and neural rough DEs [94] have demonstrated great success in the area of time-series prediction, improving the performance of NODEs, ANODEs and other deep learning models such as RNNs or LSTMs in time-series tasks. The main idea behind these models is to learn a *latent* space from the input of the neural ODE from which to sample

and predict future values. In the future, we will analyze these models in detail and explore the addition of verification techniques that can formally analyze their behavior.

V.6 Neural ODE architectures

In this section, we describe the architectures used for all the results explained in Section V.3. We utilized the same base structure to explain all GNODEs, which consist of a set of $\mathcal{NN}$-Layers, then a NODE, and then another set of $\mathcal{NN}$-Layers, except for the random experiments, which have a total of two NODEs. The names of the layers are abbreviated to include the full architecture in the tables, where layer (ns_i) corresponds to a weighted layer with activation function *layer* and ns_i neurons. If there is no number next to the layer, it means that only the activation function is applied, no weights corresponding to that layer. When a set of layers are inside brackets, it means that this subset of layers is consecutively repeated T times. For the description and architectures of the models of the fixed point attractor (FPA) and cartpole, we refer the reader to [151] and [103] respectively.

Table V.7: GNODE architectures of the spiral 2D benchmark, all the models of the damped oscillator (D.Osc.$_{n_{aug}}$), where n$_{aug} \in \{0,1,2\}$ corresponds to the number of augmented dimensions, and 6 classification models trained on MNIST.

	$\mathcal{NN}$-Layers	NODE	$\mathcal{NN}$-Layers
Spiral$_L$	N/A	fc (10) - fc (2)	N/A
Spiral$_N L$	N/A	tanh (10) - fc (2)	N/A
D.Osc.$_{n_{aug}}$	fc (2+n$_{aug}$)	fc (20) - fc (20) -fc (2+n$_{aug}$)	fc(2)
FNODE$_S$	relu (64) - relu (10)	fc (10)	softmax (10)
FNODE$_M$	relu (64) - relu (32) - fc (16)	fc (10) - fc (16)	softmax (10)
FNODE$_L$	relu (64) - [relu (32)] [x3] - fc (16)	[fc (10)] [x3] - fc (16)	softmax (10)
CNODE$_S$	[conv2d (4,3) - BN - relu] [x2] - flatten	fc (10) - fc (676)	softmax (10)
CNODE$_M$	[conv2d (10,3) - BN - relu] [x2] - flatten	fc (10) - fc (1690)	softmax (10)
CNODE$_L$	[conv2d (16,3) - BN - relu] [x2] - flatten	fc (10) - fc (2704)	softmax (10)

Table V.8: GNODE architectures of the randomly generated GNODE with multiple NODEs. Next to the model, in parentheses, is the input size of the GNODE.

	$\mathcal{NN}$-Layer	NODE	$\mathcal{NN}$-Layer	NODE	$\mathcal{NN}$-Layer
XS (1)	tanh(2)	tanh(2) - tanh(2)	tanh(2)	tanh(2)	tanh(1)
S (2)	tanh(3)	tanh(5) - tanh(3)	tanh(3)	tanh(3)	tanh(2)
M (2)	tanh(4)	tanh(8) - tanh(4)	tanh(4)	tanh(4)	tanh(2)
L (3)	tanh(4)	tanh(8) - tanh(4)	tanh(4)	tanh(4)	tanh(3)
XL (3)	tanh(5)	tanh(10) - tanh(5)	tanh(5)	tanh(5)	tanh(3)
XXL (4)	tanh(5)	tanh(10) - tanh(5)	tanh(5)	tanh(5)	tanh(4)

ACC. The ACC GNODEs used as plant models for the NNCS reachability are represented as a 3^{rd} order NODE, building upon prior knowledge of the ego and lead car dynamics. The dynamics are defined in (V.10), and the NODE architectures are described in Table V.9.

$$
\begin{aligned}
\dot{x}_1 &= \dot{x}_{\text{lead}} = v_{\text{lead}}, \\
\dot{x}_2 &= \dot{v}_{\text{lead}} = \gamma_{\text{lead}}, \\
\dot{x}_3 &= \dot{\gamma}_{\text{lead}} = \dot{y}_1, \\
\dot{x}_4 &= \dot{x}_{\text{ego}} = v_{\text{ego}}, \\
\dot{x}_5 &= \dot{v}_{\text{ego}} = \gamma_{\text{ego}}, \\
\dot{x}_6 &= \dot{\gamma}_{\text{ego}} = \dot{y}_2, \\
\dot{y} &= g_{acc}(z_{acc}), \\
z_{acc} &= [\gamma_{\text{lead}}, \gamma_{\text{ego}}, a_{\text{lead}}, a_{\text{ego}}]^\mathsf{T}
\end{aligned}
\tag{V.10}
$$

where *acc* = {L, NL}, corresponding to the linear and nonlinear NODEs.

Table V.9: Architectures of the ACC 3^{rd} order NODEs, where g_L represents the NODE architecture of the linear model and g_{NL} the nonlinear one.

	NODE
g_{NL}	tanh(10) - tanh(4)
g_L	fc(20) - fc(4)

V.7 Hyperparameters for Reachability Analysis Comparison

In this section, we describe the parameters used for each of the tools we compare against, as well as the parameters used in NNVODE. For each benchmark, we create an ordered list of parameters based on the order presented in the corresponding results table. If there is only one value for the parameters, it means that all the reachability parameters are kept constant throughout the benchmark experiments.

V.7.1 Spiral2D

Flow*

- Reach step: $\{0.01, 0.01, 0.01, 0.002, 0.002, 0.002\}$
- Taylor Models: $\{8, 8, 8, [6, 20], [6, 20], [6, 20], \}$

GoTube

- Reach step: 0.01
- Batch size: 1000

- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: 3500
- No parameters were specified for the linear model (LinearTS), we run all instances with the default parameters using $solve(problem, t_F)$

V.7.2 Fixed Point Attractor (FPA)

Flow*

- Reach step: 0.01
- Taylor Models: $\{8, 20, 30\}$

GoTube

- Reach step: 0.01
- Batch size: 1000
- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: $\{400, 1700, 3500\}$

V.7.3 Cartpole

Flow*

- Time step: 0.01
- Taylor models: $\{30, 20, 8\}$

GoTube

- Reach step: 0.01
- Batch size: 1000
- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: $\{400, 1700, 3500\}$

CHAPTER VI

Learning Nonlinear Hybrid Automata

Automata-based learning modeling of hybrid and cyber-physical systems (CPS) is a popular formal abstraction of several dynamics behaviors such as for formal verification, fault identification and anomaly detection. However, identifying hybrid systems is very challenging due to the combination of both continuous-time and discrete-time dynamics characteristic of these systems. In general, the leaning of hybrid systems has been limited to the inference of linear continuous dynamics. In this chapter, we introduce a framework to infer and verify deterministic hybrid automata with nonlinear ordinary differential equations (ODEs) from input/output execution traces. We extend an existing affine-dynamics hybrid automata learning framework and make use of its estimation of guard and transitions conditions, and its automata mode merging approach to deliver a hybrid system with nonlinear continuous dynamics. We incorporate an occupational kernel system identification method and a learning approach for neural ordinary differential equations to represent the nonlinear dynamics of the hybrid systems. We demonstrate the capabilities of the proposed methods on four benchmarks and compare to a state-of-the-art approach implemented in the extended framework.

VI.1 Introduction

Hybrid automata learning has been an active area of research for several years, but it remains a challenge in the community due to its complexity in nature. There have been several attempts to learn linear hybrid automata, but no work can successfully solve the nonlinear hybrid automata learning problem in general. This is in part due to both, the challenges in hybrid automata learning and nonlinear system identification. Simply put, there is not a modal nonlinear system identification due to this complexity yet [157]. Although there have been several methods tackling this problem such as neural networks [157], NARMAX methods [158], Lyapunov methods [159] and dynamic mode decomposition [160, 161, 162], the complexities and various ways nonlinearities may appear, are a current challenge preventing general nonlinear system identification algorithms from being available. Due to the complexities of both hybrid automata learning and nonlinear system identification, and despite the recent efforts to address some of these challenges in both areas, the combination of these two has not comprehensively studied to this day.

For this reason, we will be extending the hybrid automata learning framework introduced by Xiaodong et al. [8] with the Occupation Kernel Method for Nonlinear System Identification developed by Rosenfeld et al. [9] and with the use of neural ordinary differential equations (neural ODEs) [6]. The first method differentiates itself from previous kernel methods by separating the basis functions from data integration.

Typical kernel based approaches leverage the representer theorem to yield an approximation of the dynamics with respect to a linear combination of kernel functions centered at the data points. On the other hand, the method presented in this manuscript leverages the form of the occupation kernel to incorporate the trajectory inside a Reproducing Kernel Hilbert Space (RKHS), and the selection of occupation kernel is largely independent of the selection of basis functions. In the present context, the selection of basis functions is such that the functions should give a densely defined Liouville operator in the selected kernel space to guarantee the soundness of the developed methods. The advantage gained in the use of occupation kernels is that the occupation kernel itself is not as strongly influenced by noisy measurements as the kernel counterparts, since it can be represented as the integral of kernels that have centers along the trajectory. The second approach uses neural ordinary differential equations to estimate the continuous-time dynamics of the hybrid systems.

In summary, the contributions of this chapter are:

- Extend an existing hybrid automata learning framework to allow the learning of nonlinear continuous dynamics.
- Propose two new methods to learn the continuous dynamics of hybrid systems, using the occupational kernel system identification method and using neural ordinary differential equations.
- Evaluate the proposed methods and compare to the existing linear approaches already implemented in the extended framework.

VI.2 Background and Problem Formulation

Hybrid automaton is a formal model for dynamical systems that combine continuous-time dynamics with discrete transitions between different dynamical operation modes. Our work focuses on deterministic and synchronous models, where all constraints over state variables are specified using linear (affine) equations or inequalities. Given a set of time series traces that are generated from a hybrid system, a formal inference model, HA, for this hybrid system is inferred as a hybrid automaton. We assume the system dynamics in each mode are characterized by a nonlinear ODE:

$$\dot{\mathbf{x}} = f(\theta_q, \mathbf{x}, \mathbf{u}) \tag{VI.1}$$

where θ_q are the parameters of the function f of mode q, $\mathbf{x}$ indicates the state variable vector and $\mathbf{u}$ denotes an input vector. Let's recall the Definition 1 from section I.

Definition 11 (Hybrid Automaton) *A hybrid automaton is defined as a tuple HA = (Z, z_0, X, x_0, U, inv, T, g, h, f) where:*

- $Z = \{z_1, z_2, ..., z_N\}$ *is the finite set of locations, with an initial location* $z_0 \in Z$.
- $X \subseteq \mathbb{R}^n$ *is the continuous state space with a set of initial continuous states* x_0 *such that* $x_0 \subseteq inv(z_0)$.
- $U \subseteq \mathbb{R}^n$ *is the continuous input space.*
- $inv : Z \rightarrow 2^X$ *is a mapping that assigns an invariant* $inv(z) \subseteq X$ *to each location* z. *An invariant is a property of each location that must be satisfied by all the continuous states for a given location. A state* $(z, x) \models inv(z)$ *iff* $x \in inv(z)$.
- $T \subseteq Z \times Z$ *is the set of discrete transitions between locations. A transition from* $z_i \in Z$ *to* $z_j \in Z$ *is defined as* (z_i, z_j).
- $g : T \rightarrow 2^X$ *is a guard function that associates a guard set* $g((z_i, z_j))$ *for each transition from* z_i *to* z_j, *where* $g((z_i, z_j)) \cap inv(z_i) \neq \emptyset$.
- $h : T \times X \rightarrow X$ *is the reset function that returns the next continuous state when a transition is taken. The reset function of a transition* $g((z_i, z_j))$ *is denoted as* $h((z_i, z_j))$.
- $f : Z \times X \times U \rightarrow \mathbb{R}^n$ *is the set of ordinary differential equations (ODEs) that define the continuous dynamics for each location* $z \in Z$ *over the continuous state variables* $x \in X$.

The invariants $inv(z)$, *guard sets* $g((z_i, z_j))$ *and reset functions* $h((z_i, z_j))$ *are generally modeled as linear functions and the former two are defined as bounds over the continuous state variables. An execution of a hybrid automaton model HA is an alternating sequence of continuous flow trajectories, which evolve as defined by each location's continuous dynamics and are bounded by the invariants, and discrete transitions between these locations defined by the guard and reset functions.*

There have been several approaches to the complex problem of hybrid automata learning. In one hand, for the last two decades, there have been different methods proposed from a control and system identification perspective, on the other hand, we have the recent approaches from the deep learning and ML community using a combination of neural ODEs and other discrete functions as described in Chapter II. Despite these efforts, hybrid automata learning is still an open field of research with many challenging problems. From a system identification perspective, a common theme among most of these works and ours is that the learning process presents a similar workflow:

1. *Data Processing.* This means that the data collected to be used in the training/testing environment is processed and divided into trace segments using change-point algorithms, which then are clustered into trace segments.

2. *Model Selection.* Based on the dimensionality and type of data recorded, number of clustered data, etc., the parameters for the model are selected. Some methods allow more flexibility than others, some may learn any linear ODE models, while others select the dynamics out of a template of equations predefined.

3. *Training.* During this step, the model continuous dynamics as well as the guards conditions and transitions (discrete dynamics) are learned.

4. *Testing and Validation*. After training, it is very important to validate the system and minimize the number of nodes if possible.

Contrary to the hybrid-like neural ODEs approaches described in Chapter II, we tackle the hybrid automata learning problem from a classical system identification perspective and follow the common workflow of these approaches. The main difference is the selection of the dynamics. Given a set of trajectories collected from a real-world or simulation hybrid system, we can define the continuous time dynamics f for each mode q in two ways. The first one is using a nonlinear system identification approach based which defines the dynamics using a collection of equation of monomials of degree up to m_d [9]. The second method is with the use of neural ordinary differential equations (Neural ODEs) [6]. The methods to be used for clustering trace dynamics and learn the guards and transitions, we will make use of the framework proposed by Yang et al. [8] and extend them to the use of neural ODEs. For our framework, we will add a Step 5 to the workflow, which will include the verification of the learned hybrid systems.

VI.3 Hybrid Automata Learning Framework

In this section, we introduce the methods used to estimate and cluster the continuous-time dynamics from input-output traces, inferring guard conditions, and finally merging the modes. Our framework works as follows:

1. Given a set of trajectories or data traces, we first process the data and reduce its noise. This is followed by the changepoint estimation using the peak detection algorithm on the second-order difference of state traces, dividing the data into the possible distinct modes of the hybrid system.

2. Then, we cluster these traces using the Linear Matrix Inequality (LMI) method to detect the similarity between the trace segments, constructing a solution space.

3. The next step is to determine the guard and transitions, which we focus on inferring guards as linear inequalities (LIs), given our assumptions that the models are synchronous. We utilize the Random Sample Consensus (RANSAC) to estimate these LIs.

4. Then, we perform the system identification algorithm: either the occupational kernel method or neural ordinary differential equations.

5. Finally, we merge the modes using the prefix tree acceptor (PTA) and the hybrid automata is estimated, generating the models in Stateflow, SpaceEx and CORA formats.

In the following sections, we introduce the high-level details for each step, and refer the reader to [8] for the exact details of the hybrid automata learning framework extended, with the exception of the mode dynamics learning, which we explain the occupational learning method as well as the neural ODE one.

VI.3.1 Changepoint Estimation

The dynamics of hybrid automata are exhibited in the traces of data collected from hardware or simulation experiments. These traces are a set of finite sequences of input and output values, with a constant sampling interval. Although the data is not always collected with a fixed time-step t_s, we make this assumption in our work, with the objective of using this for both system identification routines.However, in the case of the neural ODE approach, this is not necessary. We denote a trace or trajectory of the system as $\mathcal{X}$, which is a sequence of sampled values of the outputs or state variables of the system over a time T. For detecting all the possible changes in dynamics over this time T, a changepoint is characterized by an abrupt change in value, which we calculate using the second-order derivative of the data. We define every changepoint as a timestamp τ_i, $_{i=0,1,\ldots,f}$, where $\tau_0 = 0$, $\tau_i < \tau_{i+i}$, and $\tau_f = \mathrm{T}$. With the collection of changepoints, we now have a set of trace segments $\mathcal{X}_i$, whose input-output values extend from τ_i to τ_{i+1}.

All traces in our framework are automatically segmented by applying the peak detection algorithm. While the goal is to have trace segments, it is not guaranteed that a perfect trace segmentation is achieved due to noise in the data. To overcome this problem, we use the LMI method for clustering segments to filter out erroneous segments containing multiple dynamics. LMI filters out these erroneous segments due to the overlapping of solutions, which one with multiple dynamics has very few changes of overlapping with one with single dynamics.

VI.3.2 Solution Spaces

Given a set of changepoints and its corresponding trace segments, we can compute a solution space for each trace, and then cluster these using LMIs. First, we create a solution space for each trace segment, which is the space that contains all possible dynamics, represented by a LMI. Based on this, we can check the overlapping of solution spaces to evaluate the similarity between the dynamics of the trace segments and cluster the similar segments together. This is performed by merging the trace segments into one LMI and compute its feasibility.

Two trace segments have similar dynamics if the solution of the LMI is feasible, otherwise the two traces are not considered to be similar.

To perform the clustering of the trace segments, we recursively search for all other trace segments and compute its intersection to check if they belong to the same cluster. We assume that the longer the trace segment is, the higher likelihood of encoding more dynamic information. Thus, trace segments are sorted in decreasing order and the longest segment is used as a reference for the rest. Once the clustering is performed and traces are divided into c_m clusters, we denote every trace in one cluster as the same ODE with a label $\mathcal{D}_i$, where $i \in [0,c_m]$. Then, we compute all possible transitions to later estimate the guard conditions between each cluster or mode.

VI.3.3 Mode Dynamics

After clustering the trace segments into the all the possible distinct dynamics, we are going to estimate the dynamics of each cluster with 1) the occupational kernel nonlinear method [9] and 2) neural ODEs [6, 84].

VI.3.3.1 Occupational Kernel for Nonlinear System Identification

The first method uses a monomial basis to represent the dynamics of each mode. This system identification approach makes use of Liouville operators and occupation kernels over Reproducing Kernel Hilbert Spaces (RKHS). This method integrates Liouville operators with RKHSs to compute a representation of the dynamics of each mode in a Hilbert space setting. Liouville operators are densely defined operators whose adjoint contains occupation kernels corresponding to solutions of differential equations within its domain. Hence, a dynamical system may be embedded into a RKHS where methods of numerical analysis, machine learning, and approximation theory affiliated with RKHSs may be brought to bear on problems in dynamical systems theory. The domain of Liouville operators depends on the selection of RKHS. It was demonstrated that Liouville operators with polynomial symbols are densely defined over the RKHS corresponding to the exponential dot product kernel function. Moreover, it was demonstrated in the system identification routine that the selection of kernel function may have an effect on the results of parameter estimation.

From a user's standpoint, there are a few parameters we need to choose: 1) kernel, 2) kernel width, 3) numerical integration scheme, and 4) Polynomial Degree. For the first one, we have two options: Gaussian Radial Basis Functions (RBFs) and exponential dot products, and we select the Gaussian RBFs as the default kernel in our framework. For the second one, the kernel width depends on the kernel chosen, and we estimate it based on the data to estimate and the selected kernel. For 3), we use the Simpson's rule as the integration method, as it proved to produce the most accurate results [9]. And for the final parameter, the Polynomial Degree, we typically use 2 or 3, but we leave it up to the user to select the value, as it depends more on

the data to estimate. This value stands for the maximal degree of monomials to express the dynamics of the system. For a more detailed explanation on Liouville operators, Reproducing Kernel Hilbert Spaces and the system identification approach, we refer the reader to [9].

The main outcome from this method is that we can express the dynamics in the form of monomial coefficients, which can represent both linear or nonlinear dynamics. As an example, consider we are trying to estimate a system of differential equations with two variables, and we select the default parameters from the previous paragraph and a maximum polynomial degree of 3. Then, our method will produce a vector of coefficients with size 1×10 for each state variable, 2 in our example, which corresponds to the following monomials

$$[1,\ x_1,\ x_1^2,\ x_1^3,\ x_2,\ x_1x_2,\ x_1^2x_2,\ x_2^2,\ x_1x_2^2,\ x_2^3]^\top.$$

VI.3.3.2 Neural Ordinary Differential Equations

In this work, we only make use of NODEs rather than the more generalized form proposed in Chapter V. Throughout this chapter, when we refer to the use of neural ODEs, we are making a reference to the NODEs, which are described in Definition 3 and Equations V.4 and V.1. In summary, we are defining the differential equations of the continuous dynamics of each mode with a neural network, in which we can customize: 1) the number of hidden layers, 2) number of neurons in each layer, and 3) the activation functions. For the number of layers, we require a minimum of 1 hidden layer and one output layer. In terms of the number of neurons per layer, there are no requirements, but based on our experience, 10-30 neurons works well, which we provide a default value of 20. Lastly, for the activation functions, the default value is the hyperbolic tangent function (tanh), but we also allowed the use of linear layers and sigmoid layers. For the training of all these models, we choose the ADAM algorithm [163] with an initial learning rate of 0.001 and betas of 0.9 and 0.999.

As an example, let's recall the neural ODE implementation example from Chapter V, Eq. V.8. This example models a 2-layer neural ODE with 5 neurons in the hidden layer, a hyperbolic activation function (tanh), and 2 state variables (z_1, z_2), which we write as follows

$$\begin{aligned}
\dot{z}_1 &= w_{2_{11}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{12}} tanh(w_{1_{21}} z_1 + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{13}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\
&\quad + w_{1_{14}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) + w_{2_{15}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1}, \\
\dot{z}_2 &= w_{2_{21}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{22}} tanh(w_{1_{21}} z_1 + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{23}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\
&\quad + w_{1_{24}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) + w_{2_{25}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1},
\end{aligned}$$

where the w and b terms correspond to the weight and bias parameters of the neural ODE as described in Eqs. V.6 and V.7 from Chapter V.

VI.3.4 Guard Conditions

In hybrid automata, there are two types of guard conditions: based on external events or on state variable inequalities. In our work, we consider the models to be synchronous, which means there is no delay between the input and the output, therefore, we only consider guard conditions based on state variables as the possible event's impact is immediate and will be reflected on the state variables of the system. We model these as linear inequalities (LIs), although nonlinear inequalities are also possible. Given the set of changepoints and trace segments, we estimate a LI for each possible transition computed on VI.3.2. We use an affine-subspace clustering method to estimate the LIs that clusters data into multiple low-dimensional planes. We utilize the Random Sample Consensus (RANSAC), a statistical method for estimating parameters by iteratively and randomly sampling observed data. With the RANSAC method, we compute a set of planes with a tolerance error, and the one generated with the most numbers of inliers corresponds to the most optimal one. Then, depending on the number of points to each side of the plane, the sign of the LI condition is determined. These step generates a set of all possible or preliminary transitions, defined by the segment id of the source and destination mode, the ODE-label of the source and destination modes, and the guard conditions. Using these variables, we compare the similarity between all of them, and merge and reduce the number of transitions.

VI.3.5 Merging Modes

We have estimated all possible mode dynamics, guards and transitions, so the last step is to merge the similar ODE-label traces from the previous section that correspond to the same mode dynamics, and infer the final hybrid system. For this step, we make use of the prefix tree acceptor (PTA) [8], which makes use of the preliminary transitions estimated in the previous step, in combination with the identified mode dynamics to compute the merges. Unlike previous works [23, 164] that merges two modes solely based on the probability of staying in or transitioning out of a mode, our method computes the similarity of the two modes using the estimated dynamics and transitions conditions. The core idea of the merging process is to evaluate the compatibility of the preliminary transitions, and merge the compatible ones and discard the erroneous ones, by requiring a minimum of times these transitions appear, i.e., there are two transitions from mode 1 to mode 2, each with a different guard condition, but one appears 50 times in the collected trajectories, while the second one has only one occurrence, we keep the first one and discard the second one, making our approach more robust to outliers in the data.

Once the modes are merged, we generate three different files: 1) Simulink/Stateflow, 2) SpaceEx format

and 3) CORA format. The first one is a common modelling tool provided by MathWorks for dynamical systems, typically used in the CPS community [165, 166, 167]. The last 2 formats are for two formal verification tools in the hybrid systems community [67, 65]. We perform these transformations with the use of HyST [168], a source transformation and translation tool for hybrid automaton models, as well as the built-in functions in CORA [65].

VI.4 Evaluation

Although the methods introduced in this work are designed for hybrid automata with nonlinear dynamics, we compare to the existing methods in the HAutLearn framework developed in [8], and extended in this work. We evaluate the presented methods on a set of 4 benchmarks, one with linear dynamics and the other 3 with nonlinear continuous dynamics. For all the benchmarks, we introduce the original hybrid automata, describe the data generated with these, which is later used on our framework to learn the new hybrid automata models, and present the results from each method and how each of them perform with respect to one another. A brief summary of the results is included in Table VI.1 where the test MSE values of the learned models are introduced along the computation times to estimate the continuous dynamics for all modes in the hybrid system.

Table VI.1: Test MSE values for the generated hybrid automata using the existing linear method (*linear*), the nonlinear occupational kernel system ID (*nonlinear*), and neural ordinary differential equations (*neural ODE*). The presented times correspond to the estimation of the continuous-time dynamics only.

	Example 2D linear		Example 2D nonlinear		Stable 2D		Stable 3D	
	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)
Linear	0.3339	0.6946	0.1089	0.0911	0.9049	0.1126	0.0361	0.0830
Nonlinear	0.3344	0.5112	0.0219	3.1314	0.0003	3.6347	0.0024	79.6808
Neural ODE	0.8427	131.21	0.1437	807.67	0.2186	2129.4	0.0308	5920.9

2-dimensional linear system

The first one is a simple model with 2 modes, 2 state variables and 2 transitions defined with guards defined as linear inequalities. This is the only hybrid automata model considered in this work that at least one of the continuous-time derivatives is not defined by a nonlinear equation. We can observe this hybrid automata model in Figure VI.1.

We first model this system in Simulink and generate 25 trajectories for 50 seconds with a sample time of 0.05 seconds with random initial states sampled from $x_0 \in [0, 0.1]$, $y_0 \in [0, 0.1]$, and initial location in Mode 1. Then, we use 10 of those trajectories to estimate the hybrid automata models, and test it using one of the remaining trajectories with initial state

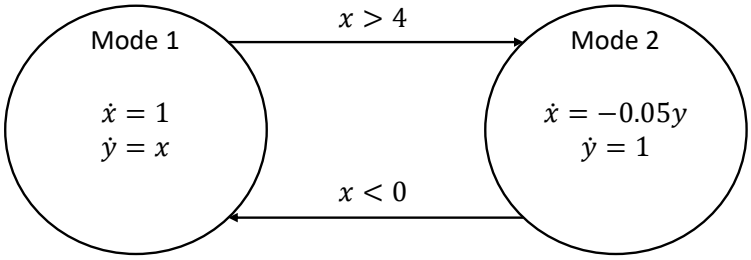
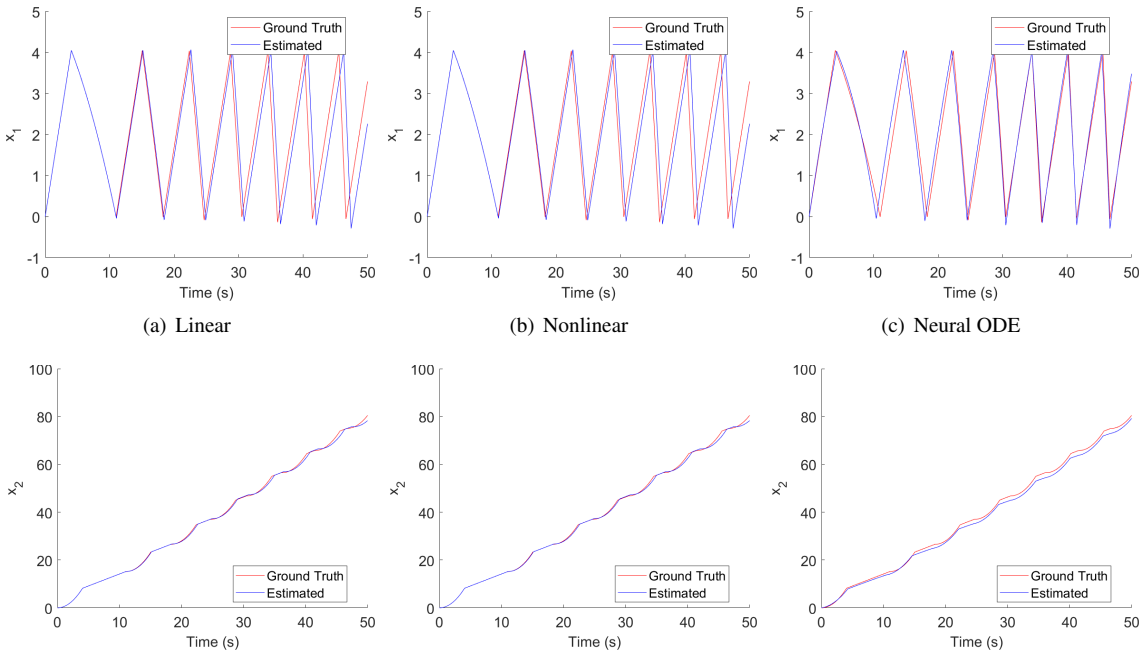


Figure VI.1: 2-dimensional hybrid automata model with linear continuous dynamics.

$$x_0 = 0 \ , \ y_0 = 0 \ , \ location = Mode\ 1.$$

The testing results are depicted in Figure VI.2.



(a) Linear (b) Nonlinear (c) Neural ODE

(d) Linear (e) Nonlinear (f) Neural ODE

Figure VI.2: Learning results of the 2-dimensional linear example.

2-dimensional nonlinear system

The next model has 2 modes, 2 state variables and 2 transitions defined with guards defined as linear inequalities as well. We can observe this hybrid automata model in Figure VI.3.

Following the same steps as in the previous example, we model this system in Simulink and generate 25 trajectories for 20 seconds with a sample time of 0.01 seconds with random initial states sampled from $x_0 \in$

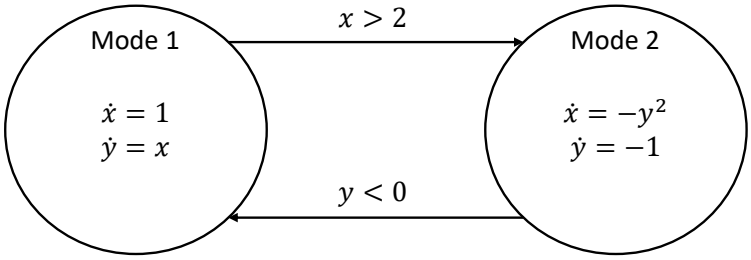
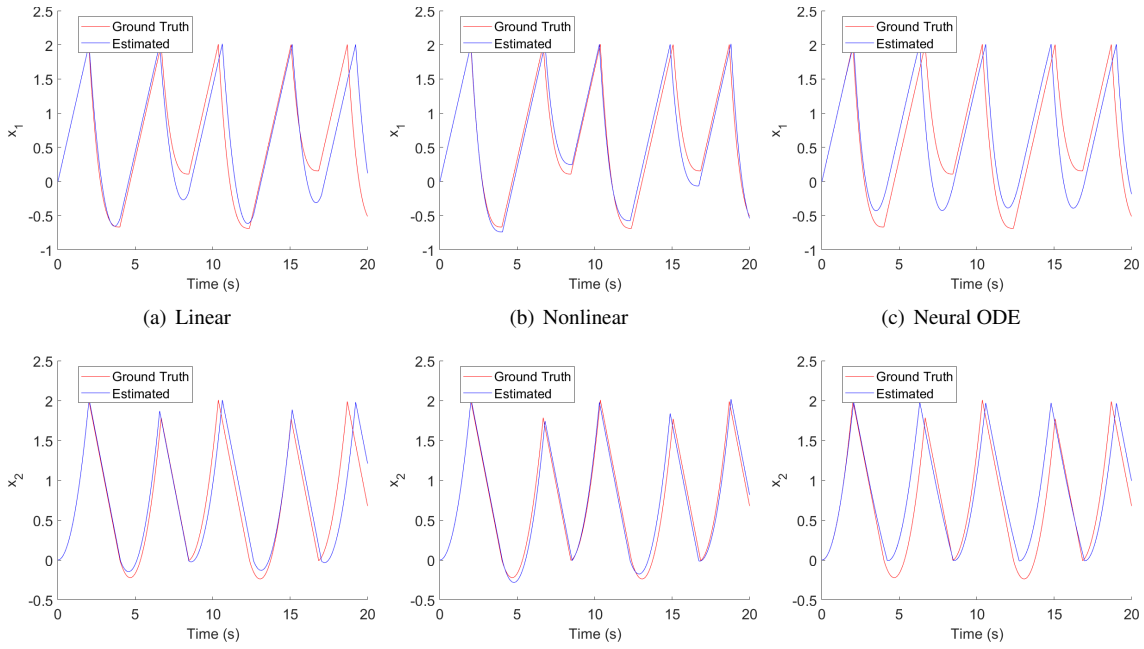


Figure VI.3: 2-dimensional hybrid automata model with nonlinear continuous dynamics.

[0, 0.1], $y_0 \in [0, 0.1]$, and initial location in Mode 1. Then, we use 10 of those trajectories to estimate the hybrid automata models, and test it using one of the remaining trajectories with initial state

$$x_0 = 0 \ , \ y_0 = 0 \ , \ location = Mode\ 1.$$

The testing results are depicted in Figure VI.4.



(a) Linear (b) Nonlinear (c) Neural ODE

(d) Linear (e) Nonlinear (f) Neural ODE

Figure VI.4: Learning results of the 2-dimensional nonlinear example.

2-dimensional stable system

This model was previously introduced and evaluated in [74] and it is depicted in Figure VI.5. The continuous dynamics in *Mode 1* are unstable, but the whole hybrid system is stable.

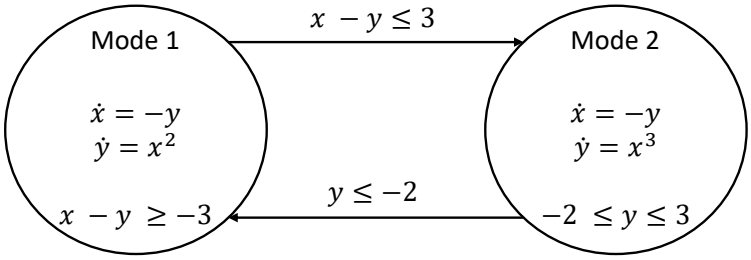
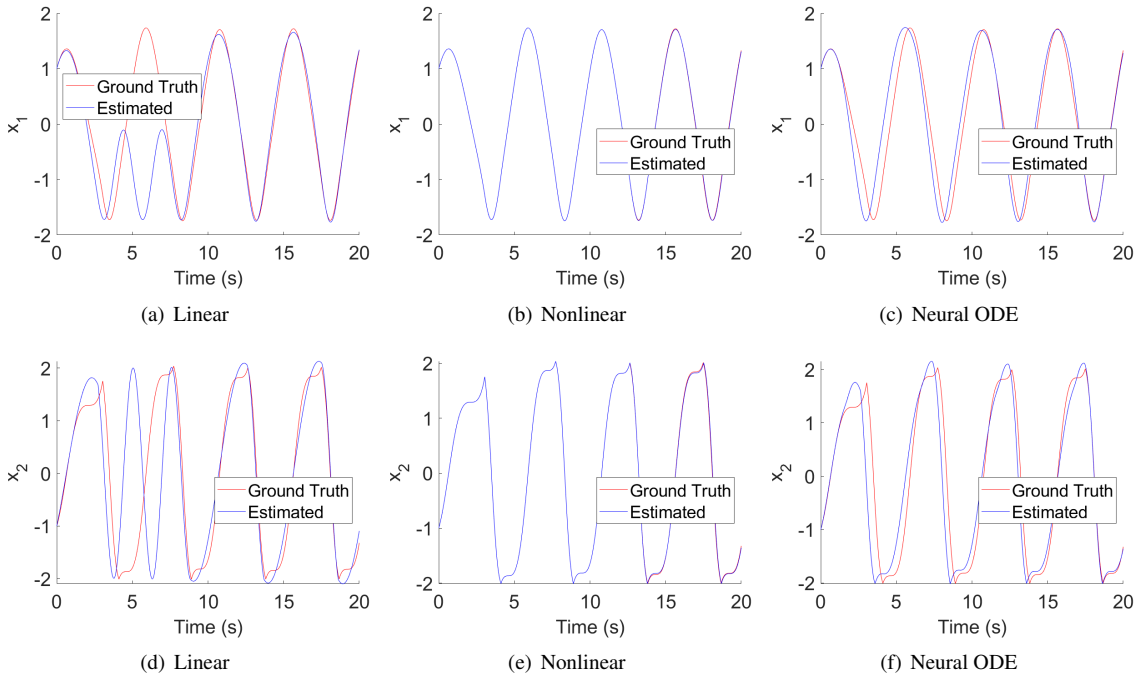


Figure VI.5: 2-dimensional hybrid automata model with nonlinear continuous dynamics.

Following the same steps as in the previous examples, we model this system in Simulink and generate 25 trajectories for 20 seconds with a sample time of 0.01 seconds with random initial states sampled from $x_0 \in [0.9, 1.1]$, $y_0 \in [-1.1, -0.9]$, and initial location in Mode 1. Then, we use 10 of those trajectories to estimate the hybrid automata models, and test it using one of the remaining trajectories with initial state

$$x_0 = 1 \ , \ y_0 = -1 \ , \ location = Mode\ 1.$$

The testing results are depicted in Figure VI.6.



(a) Linear (b) Nonlinear (c) Neural ODE

(d) Linear (e) Nonlinear (f) Neural ODE

Figure VI.6: Learning results of the 2-dimensional stable system.

3-dimensional stable system

The next model was also introduced and evaluated by Xin Chen [74], and it has 2 modes, 3 state variables and one transitions from mode 1 to mode 2. The continuous dynamics of *Mode 1* are given by

$$\begin{aligned}
\dot{x} &= -9(x-2)-7(y+2)+(z-1)+0.2(x-2)(y+2)+0.1(y+2)(z-1)+0.1(x-2)(z-1)+0.5(z-1)^2,\\
\dot{y} &= 6(x-2)+4(y+2)+(z-1),\\
\dot{z} &= 3(x-2)+2(y+2)-2.5(z-1),
\end{aligned} \tag{VI.2}$$

and the ones in *Mode 2* are given by

$$\begin{aligned}
\dot{x} &= 2.2x+3.6y+3.9z,\\
\dot{y} &= 3x+2.4y+3.4z-0.01x^2,\\
\dot{z} &= -5x-5.4y-6.7z.
\end{aligned} \tag{VI.3}$$

The only transition on this model is from *Mode 1* to *Mode 2*, whose guard condition is defined as a 3-dimensional box:

$$E = \{(x,y,z) \mid x \in [1.7,2.3] \wedge y \in [-2.3,-1.7] \wedge z \in [0.7,1.3]\}.$$

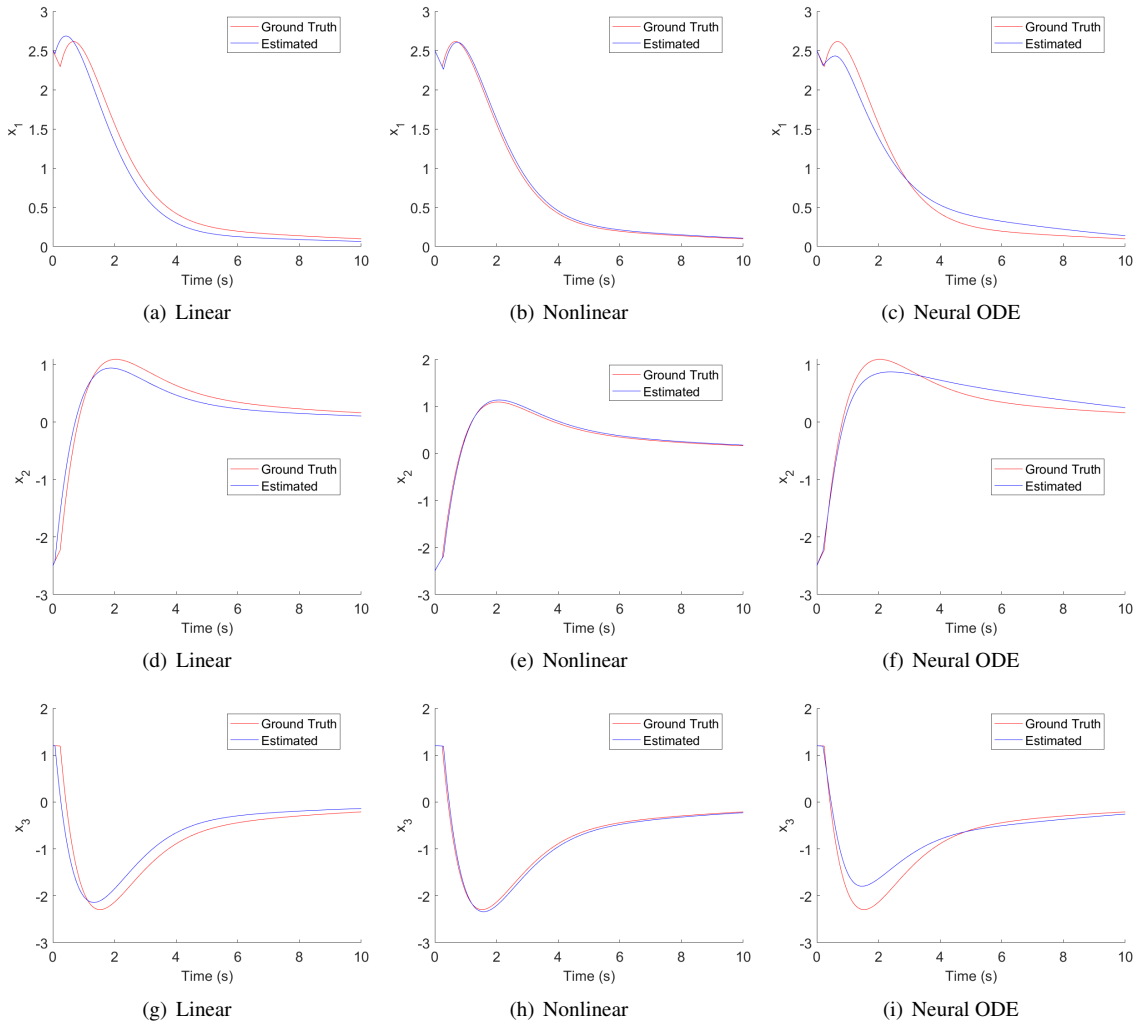
Following the same steps as in the previous example, we model this system in Simulink and generate 25 trajectories for 10 seconds with a sample time of 0.005 seconds with random initial states sampled from $x_0 \in$ [5.5, 6.5], $y_0 \in$ [-6.5, -5.5], $z_0 \in$ [3.1, 4.1], and initial location in Mode 1. Then, we use 10 of those trajectories to estimate the hybrid automata models, and select a test trajectory from outside the training distribution with initial state

$$x_0 = 2.5 \ , \ y_0 = -2.5 \ , \ z_0 = 1.2 \ , \ location = Mode\ 1.$$

The testing results are depicted in Figure VI.7.

VI.5 Discussion

There are some general trends that we can observe from Table VI.1. The first one is that the linear estimation is the fastest to compute, as expected, followed by the occupational kernel system ID, and the longest one is



(a) Linear (b) Nonlinear (c) Neural ODE

(d) Linear (e) Nonlinear (f) Neural ODE

(g) Linear (h) Nonlinear (i) Neural ODE

Figure VI.7: Learning results of the 3-dimensional stable system.

the neural ODE method, which is several orders of magnitude slower than the other two methods. Another interesting note in terms of computation time is the increase in time to estimate the nonlinear dynamics for the 3-dimensional stable system. Increasing the number of state variables from 2 to 3, increases the computation time from around 3 seconds to 80 seconds for the occupational kernel system ID method, while, surprisingly, the first and simplest model to learn is the one that takes the longer to compute for the linear method.

In terms of accuracy, the best performing method is the occupational kernel system ID method, as it has the lowest MSE values across all benchmarks, except for the first one (2-D linear example), which has approximately the same MSE as the linear method. On the other hand, the neural ODE method does not always outperform the linear one, as one may have predicted. One of the main drawbacks from this method is that the learning is heavily dependent on the initialization of the parameters, as the training may get stuck

in local minima. Here we simply introduce examples of the hybrid automata learning inference with neural ODEs, but an ablation study of the neural ODE approach would provide a better comparison, as it would reduce the randomness generated with the initialization of the parameters.

VI.6 Conclusion

Modeling complex behaviors of cyber-physical systems is an important challenge, and hybrid automata learning is one approach to overcome this. However, hybrid automata learning is very challenging due to the need to identify the continuous dynamics of the state variables, as well as the discrete behaviors and transitions between the different modes, with their corresponding guard conditions, and reset functions. In this chapter, we have extended a hybrid automata learning framework [8] and incorporated two nonlinear system identification methods to estimate the continuous dynamics from input-output traces: the occupational kernel system identification method [9] and neural ordinary differential equations [6]. We demonstrate the capabilities of our methods in a set of benchmarks and compare them to a state-of-the-art learning approach that makes use of linear ODEs to represent the continuous behavior of the systems. In the future, we will add an ablation study of the neural ODE learning method to provide with a more comprehensive evaluation of our methods. In this study, we will evaluate these methods on real-world data such as [169] as well, which will also contribute to provide a more comprehensive evaluation of our methods. Finally, we plan to develop a more automated hybrid automata learning tool by incorporating a grid search for the hyperparameter tuning needed to estimate the hybrid systems.

CHAPTER VII

Conclusions

Cyber-physical systems (CPSs) with learning-enable components (LECs) are becoming very popular, especially in the area of autonomous vehicles such as unmanned aircraft, autonomous and partially autonomous cars, and underwater vehicles. However, before they can be widely adopted in these safety-critical applications, we need to ensure their design and operation are correct and safe. To ensure the behavior of CPSs we do not only need simulation, testing, or validation techniques; safety-critical CPSs require the use of formal verification to provide mathematical guarantees on its safe and correct operation. However, due to the unstructured and ever-changing environments in which they operate, in combination with the complex interactions of the CPS components, and the lack of transparency of some of its components used such as neural networks, there is a shortage of formal methods that can scale and be applied to real-world CPSs.

Bearing the above in mind, we investigate two of these main challenges: the formal modelling of the dynamics defining the interaction of the CPS with the real world, and the lack of transparency on the internal operation of the learning-enable CPSs. For the first challenge, we have introduced a system identification approach for nonlinear input-output traces. This consists of a hybrid automata learning framework that has been extended and generalized to include two approaches: one with neural ordinary differential equations [6] and the occupational kernel system identification approach [9]. We have demonstrated the learning capabilities of our approaches on a set of four benchmarks, showing how this framework can successfully learn hybrid systems with nonlinear continuous dynamics.

For the second challenge, we can divide the work into two subcategories, first, two case studies for safety verification of autonomous vehicles, and the second one considering the reachability analysis of neural ordinary differential equations. These case studies evaluate the complexity of more realistic autonomous CPSs controlled by one or more neural networks. The first one considers the safety of an unmanned underwater vehicle (UUV), which is an NNCS composed of the vehicle dynamics, and three neural networks. The neural network controller is generated using reinforcement learning (RL) and the scenarios to generate the data is conducted using a Gazebo [111] simulator named UUVSim [112]. To formally analyze the system, we implement the approach using the NNV tool, and successfully verify the safety of the UUV under four scenarios, given an initial uncertainty on the location of the vehicle.

Similar to the first case study, we consider the safety of an aircraft controlled by the neural network compression of the Avoidance Collision Avoidance System X (ACAS X) for unmanned aircraft, when in the presence of an intruder aircraft. In this chapter, we first introduce the NNCS benchmark along with the 10

closed loop properties to verify. This benchmark incorporates several challenges, including the nonlinear 9-dimensional plant dynamics and the switching classification-based neural network controllers. In addition to the complexities, we consider an initial uncertainty in the position of the ownship aircraft of $\pm$ 5000 ft in the x-axis and $\pm$ 200 ft in the y-axis for all 10 closed loop properties to verify. To achieve this, we extended NNV to compute partitions on the initial set of the NNCS, as well as an algorithm to handle the switching behavior of the neural network controllers. With these implementations and the given assumptions, such as the intruder never changes its trajectory and no vertical movements are allowed, we are able to guarantee the safety of the ownship aircraft under all scenarios in the presence of an intruder. This study showed that it is possible to do a comprehensive safety verification analysis of a complex air collision advisory system for aircraft travelling in straight, co-altitude paths, but further research is needed to determine if climbing or descending flights can also be proven safe.

Lastly, we introduce a verification framework to compute the reachable sets of a general class of neural ODEs (GNODE) that no other existing methods are able to verify. We first introduce the GNODE architecture, which is able to encode the originally proposed neural ODE [6], as well as several of its variants, including ANODEs [84] and SONODEs [87], then the verification framework implemented using NNV [14], and finally the evaluation of our methods. In the evaluation, we demonstrate through a comprehensive set of experiments the capabilities of our methods on dynamical systems, control systems and classification tasks, as well as the comparison to state-of-the-art reachability analysis tools for continuous-time dynamical systems (NODE). However, there are several challenges in the present and future of the formal verification of neural ordinary differential equations, including the scalability of the nonlinear ODE reachability analysis as the dimension complexity of the models increased, as observed in the cartpole and damped oscillator examples, and the amount of variations and architectures that we are not able to verify, but show great promise in the area of time-series prediction. These include latent neural ODEs [12] and some of its proposed variations such as controlled neural DEs [93] and neural rough DEs [94] which improve the performance of NODEs, ANODEs and other deep learning models such as RNNs or LSTMs in time-series tasks.

In summary, the main contributions of this manuscript aim to help extend the current use of autonomous cyber-physical systems in our day-to-day lives by providing a more accurate and general hybrid automata learning framework, formally verifying the safety of two complex autonomous vehicles in underwater and aerospace operations, and providing a new framework to verify neural ordinary differential equations. We extend the state-of-the-art verification tool NNV to include the formal verification procedures of the unmanned underwater vehicle neural network control system and the formal verification of the ACAS Xu neural network compression system for two aircraft. We also extend NNV to include verification approaches for a general class of neural ordinary differential equations, and demonstrate the efficacy of our methods in sev-

eral benchmarks in the areas of control systems, dynamical systems and image recognition. In our growing world of ever-increasing dependency of embedded and cyber-physical systems, this work is an effort to contribute to the correct and safe modelling, validation and verification of autonomous systems in safety-critical applications.

VII.1 Future Work

The area of formal methods for complex learning-enabled CPS has vastly grown in the past few decades. However, there are many remaining challenges, even growing as more of these systems evolve and increase in complexity. One such case is the use of multiple neural network components on autonomous vehicles to preform safety critical tasks such as driving in a highway or navigating traffic within the city roads. The deep learning models used in these applications match the complexity of the problems they solve, intensifying the difficulty to verify these systems, a task needed to safely deploy these systems in safety-critical missions.

We observe some of these challenges in the first two chapters of this dissertation, where we verify the safety of two autonomous vehicles. Taking a closer look at the second one, the ownship aircraft commanded by the ACAS Xu control system, we observe some scalability problems that the verification methods suffer from. Although the two tools utilized successfully analyzed all the proposed scenarios, we can observe that this is achieved by a careful selection of partitioning of the initial set, which was selected after a trail-error analysis. When trying to solve the problem as a whole, our NNV methods suffer from out of memory errors due to the large initial state considered and the complexity of the nonlinear dynamical plant. We also observe, that after this partitioning, the computation time for each scenario is in average ?????? seconds. In the future, to fasten the reachable set computation of these complex NNCS, we plan to incorporate an automated partitioning regime based on the output sets of the control commands as well as automating the parallel execution of these partitions.

In the area of CPS learning and explainable AI, we plan to extend the current work on hybrid automata learning to the use of neural network as *soft* hybrid automata. The main idea behind this project is to identify, using similar clustering techniques as in Chapter VI, the active neurons during the execution traces of these models, the neurons that present the largest impact on the output of the neural network. This will yield a neural network with multiple *modes of operation*, where only a subset of neurons is *active* at different executions. This project will ease the understanding of these opaque models and visualize what parts of the system may be leading to incorrect or unsafe executions, while simplifying some of the complex learning aspects of hybrid systems.

One more question that I would like to provide some insights to is the utility of neural ordinary differential equations. In this work, we have provided a framework for the verification of a general class of neural ODEs

using reachability analysis. However, how does the verification of these models compare to other state-aware deep learning models such as Recurrent Neural Networks (RNNs)? Are neural ODEs better at learning dynamical systems than classical system identification approaches, or other deep learning approaches such as LSTMs or RNNs?

CHAPTER VIII

List of Publications

Under Review

1. Diego Manzanas Lopez, Taylor T. Johnson, Stanley Bak, Hoang-Dung Tran, Kerianne L. Hobbs, Evaluation of Neural Network Verification Methods forAir to Air Collision Avoidance In AIAA Journal of Air Transportation (Submitted March 2021 - Recommended for Publication)

Accepted Journal, Conference, and Workshop Papers

1. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton and Taylor T. Johnson, Reachability Analysis of a General Class of Neural Ordinary Differential Equations, In 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2022 September.

2. Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston Robinette, and Taylor T. Johnson, An Empirical Analysis of the Use of Real-Time Reachability for the Safety Assurance of Autonomous Vehicles, arXiv, 2022 May.

3. Nathaniel Hamilton, Patrick Musau, Diego Manzanas Lopez, Taylor Johnson, Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning, In IEEE International Conference on Assured Autonomy (ICAA), 2022, March.

4. Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston Robinette, Taylor Johnson, On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers, In IEEE International Conference on Assured Autonomy (ICAA), 2022, March.

5. Taylor T. Johnson, Diego Manzanas Lopez, Luis Benet, Marcelo Forets, Sebastián Guadalupe, Christian Schilling, Radoslav Ivanov, Taylor J. Carpenter, James Weimer and Insup Lee, ARCH-COMP21 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants In Goran Frehse and Matthias Althoff (editors). 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21), vol 80, pages 90–119. December 2021.

6. Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T Johnson, Verification of piecewise deep neural

networks: a star set approach with zonotope pre-filter, In Formal Aspects of Computing, Volume 33, Issue 4-5, pp. 519-545, 2021 August.

7. Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, Taylor T. Johnson, Robustness Verification of Semantic Segmentation Neural Networks using Relaxed Reachability, In 33rd International Conference on Computer-Aided Verification (CAV), Springer, 2021, July.

8. Diego Manzanas Lopez, Taylor T. Johnson, Hoang-Dung Tran, Stanley Bak, Xin Chen, Kerianne Hobbs, Verification of Neural Network Compression of ACAS Xu Lookup Tables with Star Set Reachability, In AIAA Scitech 2021 Forum, AIAA, 2021, January.

9. Diego Manzanas Lopez, Patrick Musau and Taylor T. Johnson On the Effectiveness of L1-Norm Based Channel Pruning for Convolutional Neural Network Verification In 2020 Verification of Neural Networks (VNN) Workshop, July 2020.

10. Taylor T. Johnson, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, Chao Huang, ARCH-COMP20 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants, In ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 74, pp. 107–139, 2020, September.

11. Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson, NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems, In 32nd International Conference on Computer-Aided Verification (CAV), 2020, July.

12. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, Hoang-Dung Tran, Taylor T. Johnson, Case Study: Safety Verification of an Unmanned Underwater Vehicle, In 2020 IEEE Security and Privacy Workshops (SPW) 2020, May

13. Hoang-Dung Tran, Diego Manzanas Lopez, Xiaodong Yang, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson, Demo: A Neural Network Verification (NNV) Tool, In 2020 IEEE Workshop on Design Automation for CPS and IoT (DESTION) 2020, April.

14. Weiming Xiang, Diego Manzanas Lopez, Patrick Musau, Taylor T. Johnson, Reachable Set Estimation and Verification for Neural Network Models of Nonlinear Dynamic Systems,In Safe, Autonomous and

Intelligent Vehicles. Unmanned System Technologies (Huafeng Yu, Xin Li, Richard M. Murray, S. Ramesh, Claire J. Tomlin, eds.), Springer International Publishing, pp. 123–144, 2019.

15. Hoang-Dung Tran, Feiyang Cai, Diego Manzanas Lopez, Patrick Musau, Taylor T. Johnson and Xenofon Koutsoukos Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control, In ACM Transactions on Embedded Computing Systems, Volume 18, Issue No. 5s, Article 105, 2019, October.

16. Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Taylor T. Johnson Star-Based Reachability Analysis for Deep Neural Networks In 23rd International Symposium on Formal Methods (FM'19) (, ed.), Springer International Publishing, 2019, October.

17. Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Taylor T. Johnson, Parallelizable Reachability Analysis Algorithms for Feed-forward Neural Networks, In Proceedings of the 7th International Workshop on Formal Methods in Software Engineering (FormaliSE'19), IEEE Press, Piscataway, NJ, USA, pp. 31–40, 2019, May.

18. Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, Taylor T. Johnson, ARCH-COMP19 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants, In ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 61, pp. 103–119, 2019, April.

19. Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Taylor T. Johnson, Verification of Closed-loop Systems with Neural Network Controllers, In ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 61, pp. 201–210, 2019, April.

20. Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, Taylor T. Johnson Verification for Machine Learning, Autonomy, and Neural Networks Survey ArXiV, 2018, October.

21. Patrick Musau, Diego Manzanas Lopez, Hoang-Dung Tran, Taylor T. Johnson, Differential Algebraic Equations (DAEs) with Varying Index (Benchmark Proposal), In 5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Oxford, UK, 2018, July.

List of Presentations

1. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton and Taylor T. Johnson, "Verification of Neural Ordinary Differential Equations using Reachability Analysis" (Poster). In 1st Workshop on Formal Verification of Machine Learning (WFVML 2022), co-located with ICML 2022.

2. Diego Manzanas Lopez, Taylor T. Johnson, Hoang-Dung Tran, Stanley Bak, Xin Chen, Kerianne Hobbs, "Verification of Neural Network Compression of ACAS Xu Lookup Tables with Star Set Reachability" (Virtual Presentation). In AIAA Scitech 2021 Forum, AIAA, 2021, January.

3. Diego Manzanas Lopez, Patrick Musau and Taylor T. Johnson "On the Effectiveness of L1-Norm Based Channel Pruning for Convolutional Neural Network Verification" (Virutal Presentation). In 2020 Verification of Neural Networks (VNN) Workshop, July 2020.

4. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, Hoang-Dung Tran, Taylor T. Johnson, "Case Study: Safety Verification of an Unmanned Underwater Vehicle" (Virtual Presentation). In 2020 IEEE Security and Privacy Workshops (SPW) 2020, May.

5. Hoang-Dung Tran, Diego Manzanas Lopez, Xiaodong Yang, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson, "Demo: A Neural Network Verification (NNV) Tool" (Virtual Demo). In 2020 IEEE Workshop on Design Automation for CPS and IoT (DESTION) 2020, April.

6. Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Taylor T. Johnson, "Verification of Closed-loop Systems with Neural Network Controllers" (Presentation). In ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 61, pp. 201–210, 2019, April.

BIBLIOGRAPHY

[1] P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, and et al., "Artificial intelligence and life in 2030.," Sep 2016.

[2] G. Karsai and J. Sztipanovits, "Model-integrated development of cyber-physical systems," in *Software Technologies for Embedded and Ubiquitous Systems* (U. Brinkschulte, T. Givargis, and S. Russo, eds.), (Berlin, Heidelberg), pp. 46–54, Springer Berlin Heidelberg, 2008.

[3] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[4] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. A. Rosenfeld, and T. T. Johnson, "Verification for machine learning, autonomy, and neural networks survey," *ArXiv preprint*, vol. abs/1810.01989, 2018.

[5] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, 2018.

[7] J. Jia and A. R. Benson, "Neural jump stochastic differential equations," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 9847–9858, Curran Associates, Inc., 2019.

[8] X. Yang, O. A. Beg, M. Kenigsberg, and T. T. Johnson, "A framework for identification and validation of affinehybrid automata from input-output traces," *preprint*, 2021.

[9] J. A. Rosenfeld, B. Russo, R. Kamalapurkar, and T. T. Johnson, "The occupation kernel method for nonlinear system identification," 2021.

[10] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[11] R. T. Q. Chen, B. Amos, and M. Nickel, "Learning neural event functions for ordinary differential equations," in *International Conference on Learning Representations*, 2021.

[12] Y. Rubanova, R. T. Q. Chen, and D. K. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[13] T. T. Johnson, D. M. Lopez, L. Benet, M. Forets, S. Guadalupe, C. Schilling, R. Ivanov, T. J. Carpenter, J. Weimer, and I. Lee, "Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)* (G. Frehse and M. Althoff, eds.), vol. 80 of *EPiC Series in Computing*, pp. 90–119, EasyChair, 2021.

[14] H.-D. Tran, X. Yang, D. M. 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 *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[15] D. M. Lopez, P. Musau, N. Hamilton, H.-D. Tran, and T. T. Johnson, "Case study: Safety verification of an unmanned underwater vehicle," in *Workshop on Assured Autonomous Systems (WAAS)*, IEEE, May 2020.

[16] D. M. Lopez, T. Johnson, H.-D. Tran, S. Bak, X. Chen, and K. L. Hobbs, *Verification of Neural Network Compression of ACAS Xu Lookup Tables with Star Set Reachability*.

[17] R. Medhat, S. Ramesh, B. Bonakdarpour, and S. Fischmeister, "A framework for mining hybrid automata from input/output traces," in *Proceedings of the 12th International Conference on Embedded Software*, pp. 177–186, IEEE Press, 2015.

[18] M. García Soto, T. A. Henzinger, C. Schilling, and L. Zeleznik, "Membership-based synthesis of linear hybrid automata," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), (Cham), pp. 297–314, Springer International Publishing, 2019.

[19] M. G. Soto, T. Henzinger, and C. Schilling, "Synthesis of hybrid automata with affine dynamics from time-series data," *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021.

[20] A. Pferscher, B. Aichernig, and M. Tappler, "From passive to active: Learning timed automata efficiently," in *12th NASA Formal Methods Symposium*, 2020.

[21] M. Tappler, B. K. Aichernig, K. G. Larsen, and F. Lorber, "Time to learn – learning timed automata from tests," in *Formal Modeling and Analysis of Timed Systems* (É. André and M. Stoelinga, eds.), pp. 216–235, Springer International Publishing, 2019.

[22] S. E. Verwer, *Efficient identification of timed automata: Theory and practice*. PhD thesis, 2010.

[23] O. Niggemann, B. Stein, A. Vodencarevic, A. Maier, and H. K. Büning, "Learning behavior models for hybrid timed systems," 2012.

[24] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal, "Identification of hybrid systems a tutorial," *European journal of control*, vol. 13, no. 2-3, pp. 242–260, 2007.

[25] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[26] A. Garulli, S. Paoletti, and A. Vicino, "A survey on switched and piecewise affine system identification," *IFAC Proceedings Volumes*, vol. 45, no. 16, pp. 344–355, 2012.

[27] L. Bako and R. Vidal, "Algebraic identification of mimo sarx models," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 43–57, Springer, 2008.

[28] Y. Ma and R. Vidal, "Identification of deterministic switched arx systems via identification of algebraic varieties," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 449–465, Springer, 2005.

[29] R. Vidal, S. Soatto, Y. Ma, and S. Sastry, "An algebraic geometric approach to the identification of a class of linear hybrid systems," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 1, pp. 167–172, IEEE, 2003.

[30] R. Vidal, "Identification of pwarx hybrid models with unknown and possibly different orders," in *Proceedings of the 2004 American Control Conference*, vol. 1, pp. 547–552, IEEE, 2004.

[31] R. Vidal, "Recursive identification of switched arx systems," *Automatica*, vol. 44, no. 9, pp. 2274–2287, 2008.

[32] R. Baptista, J. Ishihara, and G. Borges, "Split and merge algorithm for identification of piecewise affine systems," in *Proceedings of the 2011 American Control Conference*, pp. 2018–2023, IEEE, 2011.

[33] K. Boukharouba, L. Bako, and S. Lecoeuche, "Identification of piecewise affine systems based on dempster-shafer theory," *IFAC Proceedings Volumes*, vol. 42, no. 10, pp. 1662–1667, 2009.

[34] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari, "A clustering technique for the identification of piecewise affine systems," *Automatica*, vol. 39, no. 2, pp. 205–217, 2003.

[35] M. Gegundez, J. Aroba, and J. M. Bravo, "Identification of piecewise affine systems by means of fuzzy clustering and competitive learning," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 8, pp. 1321–1329, 2008.

[36] A. M. Ivanescu, T. Albin, D. Abel, and T. Seidl, "Employing correlation clustering for the identification of piecewise affine models," in *Proceedings of the 2011 workshop on Knowledge discovery, modeling and simulation*, pp. 7–14, 2011.

[37] L. Bako, "Identification of switched linear systems via sparse optimization," *Automatica*, vol. 47, no. 4, pp. 668–677, 2011.

[38] X. Jin and B. Huang, "Robust identification of piecewise/switching autoregressive exogenous process," *AIChE journal*, vol. 56, no. 7, pp. 1829–1844, 2010.

[39] C. Y. Lai, C. Xiang, and T. H. Lee, "Identification and control of nonlinear systems via piecewise affine approximation," in *49th IEEE Conference on Decision and Control (CDC)*, pp. 6395–6402, IEEE, 2010.

[40] K. Mikami, H. Okuda, S. Taguchi, Y. Tazaki, and T. Suzuki, "Model predictive assisting control of vehicle following task based on driver model," in *2010 IEEE International Conference on Control Applications*, pp. 890–895, IEEE, 2010.

[41] H. Ohlsson and L. Ljung, "Identification of piecewise affine systems using sum-of-norms regularization," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 6640–6645, 2011.

[42] R. Grosu, S. Mitra, P. Ye, E. Entcheva, I. V. Ramakrishnan, and S. A. Smolka, "Learning cycle-linear hybrid automata for excitable cells," in *Hybrid Systems: Computation and Control* (A. Bemporad, A. Bicchi, and G. Buttazzo, eds.), (Berlin, Heidelberg), pp. 245–258, Springer Berlin Heidelberg, 2007.

[43] H. Raffelt, B. Steffen, and T. Berg, "Learnlib: A library for automata learning and experimentation," in *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pp. 62–71, ACM, 2005.

[44] A. Summerville, J. Osborn, and M. Mateas, "Charda: Causal hybrid automata recovery via dynamic analysis," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, p. 2800–2806, AAAI Press, 2017.

[45] I. Lamrani, A. Banerjee, and S. K. S. Gupta, "Hymn: Mining linear hybrid automata from input output traces of cyber-physical systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 264–269, 2018.

[46] T. Sarkar, A. Rakhlin, and M. Dahleh, "Nonparametric system identification of stochastic switched linear systems," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 3623–3628, 2019.

[47] B. K. Aichernig, R. Bloem, M. Ebrahimi, M. Horn, F. Pernkopf, W. Roth, A. Rupp, M. Tappler, and M. Tranninger, "Learning a behavior model of hybrid systems through combining model-based testing and machine learning," in *Testing Software and Systems* (C. Gaston, N. Kosmatov, and P. Le Gall, eds.), (Cham), pp. 3–21, Springer International Publishing, 2019.

[48] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Learning and verification of feedback control systems using feedforward neural networks," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 151–156, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[49] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, (New York, NY, USA), pp. 157–168, ACM, 2019.

[50] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, (New York, NY, USA), pp. 169–178, ACM, 2019.

[51] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, "Verisig 2.0: Verification of neural network controllers using taylor model preconditioning," in *Computer Aided Verification* (A. Silva and K. R. M. Leino, eds.), (Cham), pp. 249–262, Springer International Publishing, 2021.

[52] 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*, HSCC '19, (New York, NY, USA), p. 147–156, Association for Computing Machinery, 2019.

[53] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," in *Safe, Autonomous and Intelligent Vehicles* (H. Yu, X. Li, R. M. Murray, S. Ramesh, and C. J. Tomlin, eds.), pp. 123–144, Springer International Publishing, 2019.

[54] W. Xiang, H.-D. Tran, J. Rosenfeld, and T. T. Johnson, "Reachable set estimation and verification for a class of piecewise linear systems with neural network controllers," in *American Control Conference (ACC 2018), Special Session on Formal Methods in Controller Synthesis I*, IEEE, June 2018.

[55] H.-D. Tran, F. Cei, D. M. Lopez, T. T. Johnson, and X. Koutsoukos, "Safety verification of cyber-physical systems with reinforcement learning control," in *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)*, ACM, October 2019.

[56] M. E. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio, "Formal verification of neural agents in non-deterministic environments," in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, IFAAMAS 2020, (Auckland, New Zealand), ACM, May 2020.

[57] C. Sidrane and M. J. Kochenderfer, "OVERT: Verification of nonlinear dynamical systems with neural network controllers via overapproximation," *Safe Machine Learning workshop at ICLR*, 2019.

[58] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu, "Reachnn*: A tool for reachability analysis of neural-network controlled systems," in *Automated Technology for Verification and Analysis* (D. V. Hung and O. Sokolsky, eds.), (Cham), pp. 537–542, Springer International Publishing, 2020.

[59] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu, "Polar: A polynomial arithmetic framework for verifying neural-network controlled systems," 2021.

[60] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Foundations and Trends in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021.

[61] H.-D. Tran, W. Xiang, and T. T. Johnson, "Verification approaches for learning-enabled autonomous cyber-physical systems," *IEEE Design and Test*, pp. 1–1, 2020.

[62] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 1821–1830, May 2021.

[63] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, Mar. 2018.

[64] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[65] M. Althoff, "An introduction to cora 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[66] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *CAV*, 2013.

[67] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV)* (S. Q. Ganesh Gopalakrishnan, ed.), LNCS, Springer, 2011.

[68] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020.

[69] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, *Output Range Analysis for Deep Feedforward Neural Networks*, pp. 121–138. 01 2018.

[70] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, "Reachnn: Reachability analysis of neural-network controlled systems," *ACM Trans. Embed. Comput. Syst.*, vol. 18, Oct. 2019.

[71] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling, "Juliareach: a toolbox for set-based reachability," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 39–44, 2019.

[72] S. Bak and P. S. Duggirala, "Simulation-equivalent reachability of large linear systems with inputs," in *Computer Aided Verification* (R. Majumdar and V. Kunčak, eds.), (Cham), pp. 401–420, Springer International Publishing, 2017.

[73] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *SP*, pp. 3–18, IEEE Computer Society, 2018.

[74] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," in *PhD thesis, RWTH Aachen University*, 2015.

[75] K. D. Julian and M. J. Kochenderfer, "A reachability method for verifying dynamical systems with deep neural network controllers," *CoRR*, vol. abs/1903.00520, 2019.

[76] T. T. Johnson, D. M. Lopez, P. Musau, H.-D. Tran, E. Botoeva, F. Leofante, A. Maleki, C. Sidrane, J. Fan, and C. Huang, "Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)* (G. Frehse and M. Althoff, eds.), vol. 74 of *EPiC Series in Computing*, pp. 107–139, EasyChair, 2020.

[77] D. M. Lopez, P. Musau, H.-D. Tran, S. Dutta, T. J. Carpenter, R. Ivanov, and T. T. Johnson, "Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems* (G. Frehse and M. Althoff, eds.), vol. 61 of *EPiC Series in Computing*, pp. 103–119, EasyChair, April 2019.

[78] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Case study: Verifying the safety of an autonomous racing car with a neural network controller," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[79] K. D. Julian and M. J. Kochenderfer, "Reachability analysis for neural network aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 0, no. 0, pp. 1–11, 2021.

[80] K. D. Julian, S. Sharma, J.-B. Jeannin, and M. J. Kochenderfer, "Verifying aircraft collision avoidance neural networks through linear approximations of safe regions," *AIAA Spring Symposium*, 2019.

[81] A. Irfan, K. D. Julian, H. Wu, C. Barrett, M. J. Kochenderfer, B. Meng, and J. Lopez, "Towards verification of neural networks for small unmanned aircraft collision avoidance," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–10, 2020.

[82] A. Clavière, E. Asselin, C. Garion, and C. Pagetti, "Safety verification of neural network controlled systems," 2020.

[83] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018.

[84] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[85] A. Gholaminejad, K. Keutzer, and G. Biros, "Anode: Unconditionally accurate memory-efficient gradients for neural odes," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 730–736, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[86] T. Zhang, Z. Yao, A. Gholami, J. E. Gonzalez, K. Keutzer, M. W. Mahoney, and G. Biros, "Anodev2: A coupled neural ode framework," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[87] A. Norcliffe, C. Bodnar, B. Day, N. Simidjievski, and P. Liò, "On second order behaviour in augmented neural odes," in *Advances in Neural Information Processing Systems*, 2020.

[88] K. Choromanski, J. Q. Davis, V. Likhosherstov, X. Song, J.-J. Slotine, J. Varley, H. Lee, A. Weller, and V. Sindhwani, "An ode to an ode," 2020.

[89] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[90] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *International Conference on Learning Representations*, 2019.

[91] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. Oberman, "How to train your neural ODE: the world of Jacobian and kinetic regularization," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 3154–3164, PMLR, 13–18 Jul 2020.

[92] A. Gholaminejad, K. Keutzer, and G. Biros, "Anode: Unconditionally accurate memory-efficient gradients for neural odes," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 730–736, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[93] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural Controlled Differential Equations for Irregular Time Series," *Advances in Neural Information Processing Systems*, 2020.

[94] J. Morrill, C. Salvi, P. Kidger, and J. Foster, "Neural rough differential equations for long time series," in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 7829–7838, PMLR, 18–24 Jul 2021.

[95] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, "Dissecting neural odes," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 3952–3963, Curran Associates, Inc., 2020.

[96] F. Carrara, R. Caldelli, F. Falchi, and G. Amato, "On the robustness to adversarial examples of neural ode image classifiers," in *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, 2019.

[97] H. Yan, J. Du, V. Y. F. Tan, and J. Feng, "On robustness of neural ordinary differential equations," 2020.

[98] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[99] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009.

[100] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 01 2011.

[101] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[102] S. Gruenbacher, R. M. Hasani, M. Lechner, J. Cyranka, S. A. Smolka, and R. Grosu, "On the verification of neural odes with stochastic guarantees," in *AAAI*, 2021.

[103] S. Gruenbacher, J. Cyranka, M. Lechner, M. A. Islam, S. A. Smolka, and R. Grosu, "Lagrangian reachtubes: The next generation," 2020.

[104] S. Gruenbacher, M. Lechner, R. Hasani, D. Rus, T. A. Henzinger, S. Smolka, and R. Grosu, "Gotube: Scalable stochastic verification of continuous-depth models," 2021.

[105] M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, and J. Park, "Graph neural ordinary differential equations," *arXiv preprint arXiv:1911.07532*, 2019.

[106] R. Shi and Q. Morris, "Segmenting hybrid trajectories using latent odes," in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 9569–9579, PMLR, 18–24 Jul 2021.

[107] M. Poli, S. Massaroli, L. Scimeca, S. J. Oh, S. Chun, A. Yamashita, H. Asama, J. Park, and A. Garg, "Neural hybrid automata: Learning dynamics with multiple modes and stochastic transitions," 2021.

[108] H.-D. Tran, P. Musau, D. M. Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis for deep neural networks," in *23rd International Symposium on Formal Methods (FM'19)*, Springer International Publishing, October 2019.

[109] J. Bellingham, "Platforms: Autonomous underwater vehicles," in *Encyclopedia of Ocean Sciences (Second Edition)* (J. H. Steele, ed.), pp. 473 – 484, Oxford: Academic Press, second edition ed., 2009.

[110] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Springer Publishing Company, Incorporated, 1st ed., 2017.

[111] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004.

[112] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, IEEE, sep 2016.

[113] L. Ljung, *System Identification (2nd Ed.): Theory for the User*. USA: Prentice Hall PTR, 1999.

[114] M. S. I. Toolbox, *(R2019b)*. Natick, Massachusetts: The MathWorks Inc., 2019.

[115] N. Indurkhya and F. J. Damerau, *Handbook of natural language processing*, vol. 2. CRC Press, 2010.

[116] D. Tuia, M. Volpi, L. Copa, M. Kanevski, and J. Munoz-Mari, "A survey of active learning algorithms for supervised remote sensing image classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 3, pp. 606–617, 2011.

[117] J. Han, D. Zhang, G. Cheng, N. Liu, and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: a survey," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 84–100, 2018.

[118] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[119] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[120] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

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

[122] M. E. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio, "Verifying Strategic Abilities of Neural-symbolic Multi-agent Systems," in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 22–32, 9 2020.

[123] S. Bak, C. Liu, and T. T. Johnson, "The second international verification of neural networks competition (VNN-COMP 2021): Summary and results," *CoRR*, vol. abs/2109.00498, 2021.

[124] T. T. Johnson, D. M. Lopez, L. Benet, M. Forets, S. Guadalupe, C. Schilling, R. Ivanov, T. J. Carpenter, J. Weimer, and I. Lee, "Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)* (G. Frehse and M. Althoff, eds.), vol. 80 of *EPiC Series in Computing*, pp. 90–119, EasyChair, 2021.

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

[126] L. E. Alvarez, I. Jessen, M. P. Owen, J. Silbermann, and P. Wood, "Acas sxu: Robust decentralized detect and avoid for small unmanned aircraft systems," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pp. 1–9, IEEE, 2019.

[127] S. Bak, "nnenum: Verification of relu neural networks with optimized abstraction refinement," in *NASA Formal Methods Symposium*, Springer, 2021.

[128] W. A. Olson, "Airborne collision avoidance system x," tech. rep., Massachusetts Institute of Technology, Lincoln Laboratory, 2015.

[129] M. J. Kochenderfer, "Optimized airborne collision avoidance," *Decision making under uncertainty: theory and application*, 2015.

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

[131] M. Marston and G. Baca, "Acas-xu initial self-separation flight tests," 2015.

[132] J. E. Holland, M. J. Kochenderfer, and W. A. Olson, "Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance," *Air Traffic Control Quarterly*, vol. 21, no. 3, pp. 275–297, 2013.

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

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

[135] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, vol. 92, no. 3, pp. 401–422, 2004.

[136] M. J. Kochenderfer and N. Monath, "Compression of optimal value functions for markov decision processes," in *2013 Data Compression Conference*, pp. 501–501, IEEE, 2013.

[137] S. Raychaudhuri, "Introduction to monte carlo simulation," in *2008 Winter Simulation Conference*, pp. 91–100, Institute of Electrical and Electronics Engineers (IEEE), 2008.

[138] J. Antony, *Design of Experiments for Engineers and Scientists*. Elsevier, 2014.

[139] F. A. Viana, "A tutorial on latin hypercube design of experiments," *Quality and Reliability Engineering International*, vol. 32, no. 5, pp. 1975–1985, 2016.

[140] *VFR cruising altitude or flight level*, 2003. 14 CFR §91.159.

[141] E. Hainry, "Reachability in linear dynamical systems," in *Logic and Theory of Algorithms* (A. Beckmann, C. Dimitracopoulos, and B. Löwe, eds.), (Berlin, Heidelberg), pp. 241–250, Springer Berlin Heidelberg, 2008.

[142] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018* (J. Lang, ed.), pp. 2651–2659, International Joint Conferences on Artificial Intelligence, July 2018.

[143] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, "Improved geometric path enumeration for verifying ReLU neural networks," in *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[144] D. Manzanas Lopez, P. Musau, N. Hamilton, and T. Johnson, "Reachability analysis of a general class of neural ordinary differential equation," in *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022), Co-Located with CONCUR, FMICS, and QEST as part of CONFEST 2022.*, (Warsaw, Poland), September 2022.

[145] D. M. Lopez, P. Musau, N. Hamilton, and T. T. Johnson, "Reachability analysis of a general class of neural ordinary differential equations," 2022.

[146] K. Hao, "A radical new neural network design could overcome big challenges in ai," Apr 2020.

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

[148] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), (Cham), pp. 443–452, Springer International Publishing, 2019.

[149] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *32nd International Conference on Computer-Aided Verification (CAV)*, Springer, July 2020.

[150] M. Althoff, "Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, (New York, NY, USA), p. 173–182, Association for Computing Machinery, 2013.

[151] P. Musau and T. T. Johnson, "Continuous-time recurrent neural networks (ctrnns) (benchmark proposal)," in *5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH)*, (Oxford, UK), july 2018.

[152] L. Doyen, G. Frehse, G. J. Pappas, and A. Platzer, "Verification of hybrid systems," in *Handbook of Model Checking* (E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, eds.), pp. 1047–1110, Springer, 2018.

[153] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, *Case Study: Verifying the Safety of an Autonomous Racing Car with a Neural Network Controller*. New York, NY, USA: Association for Computing Machinery, 2020.

[154] R. Ivanov, K. Jothimurugan, S. Hsu, S. Vaidya, R. Alur, and O. Bastani, "Compositional learning and verification of neural network controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 20, sep 2021.

[155] D. Li, S. Bak, and S. Bogomolov, "Reachability analysis of nonlinear systems using hybridization and dynamics scaling," in *International Conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '20, Springer, 2020.

[156] S. Bak, S. Bogomolov, P. S. Duggirala, A. R. Gerlach, and K. Potomkin, "Reachability of black-box nonlinear systems after koopman operator linearization," in *7th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2021, Brussels, Belgium, July 7-9, 2021* (R. M. Jungers, N. Ozay, and A. Abate, eds.), vol. 54 of *IFAC-PapersOnLine*, pp. 253–258, Elsevier, 2021.

[157] O. Nelles, *Nonlinear system identification. From classical approaches to neural networks and fuzzy models*. 01 2001.

[158] S. Billings, "Nonlinear system identification: Narmax methods in the time, frequency, and spatio-temporal domains," *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*, 08 2013.

[159] A. Parikh, R. Kamalapurkar, and W. E. Dixon, "Integral concurrent learning: Adaptive control with parameter convergence using finite excitation," *International Journal of Adaptive Control and Signal Processing*, vol. 33, no. 12, pp. 1775–1787, 2019.

[160] M. Hemati, M. O. Williams, and C. Rowley, "Dynamic mode decomposition for large and streaming datasets," *Physics of Fluids*, vol. 26, p. 111701, 2014.

[161] R. A. Taylor, J. Kutz, K. Morgan, and B. Nelson, "Dynamic mode decomposition for plasma diagnostics and validation.," *The Review of scientific instruments*, vol. 89 5, p. 053501, 2018.

[162] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[163] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[164] R. C. Carrasco and J. Oncina, "Learning deterministic regular grammars from stochastic samples in polynomial time," *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, vol. 33, no. 1, pp. 1–19, 1999.

[165] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[166] J. Lu, D. Chen, G. Wang, D. Kiritsis, and M. Törngren, "Model-based systems engineering toolchain for automated parameter value selection," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 4, pp. 2333–2347, 2022.

[167] B. van der Sanden, Y. Li, J. van den Aker, B. Akesson, T. Bijlsma, M. Hendriks, K. Triantafyllidis, J. Verriet, J. Voeten, and T. Basten, "Model-driven system-performance engineering for cyber-physical systems : Industry session paper," in *2021 International Conference on Embedded Software (EMSOFT)*, pp. 11–22, 2021.

[168] S. Bak, S. Bogomolov, and T. T. Johnson, "Hyst: A source transformation and translation tool for hybrid automaton models," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, (New York, NY, USA), p. 128–133, Association for Computing Machinery, 2015.

[169] "Nonlinear benchmarks, *Workshop on Nonlinear System Identification Benchmarks*." https://www.nonlinearbenchmark.org/benchmarks, 2022. Accessed: 2022-06-07.